Development of the marker and cell method for use with unstructured meshes.

Pelley, Rachel Elizabeth

Prifysgol Abertawe
Swansea University

EPSRC
Pioneering research and skills

SWANSEA UNIVERSITY

Development of the Marker and Cell method for use with unstructured meshes

Author:
Rachel Elizabeth PELLEY

Supervisor:
Prof. Oubay HASSAN
Prof. Kenneth MORGAN

SUBMITTED TO SWANSEA UNIVERSITY IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Year of Submission: 2013

Abstract

The marker and cell method is an efficient co-volume technique suitable for the solution of incompressible flows using a Cartesian mesh. For flows around complex geometries the use of an unstructured mesh is desirable. For geometric flexibility an unstructured mesh implementation is desirable. A co–volume technique requires a dual orthogonal mesh, in the triangular case the Delaunay-Voronoï dual provides the means for determining this dual orthogonal mesh in an unstructured mesh framework. Certain mesh criteria must be placed on the Delaunay-Voronoï to ensure it meets the dual orthogonal requirements.

The two dimensional extension of the marker and cell method to an unstructured framework is presented. The requirements of the mesh are defined and methods in their production are discussed. Initially an explicit time stepping scheme is implemented which allows efficient simulation of incompressible fluid flow problems. Limitations of the explicit time stepping scheme that were discovered, mean that high Reynolds number flows that require the use of stretched meshes cannot produce solutions in a reasonable time period. A semi-implicit time stepping routine removes this limitation allowing these types of flows to be successfully modelled.

To validate the solvers accuracy and demonstrate its performance, a number of test cases are presented. These include the lid driven cavity, flow over a backward facing step, inviscid flow around a circular cylinder, unsteady flow around a circular cylinder, flow around an SD7003 aerofoil, flow around a NACA0012 aerofoil and flow around a multi element aerofoil.

The investigation although revealing a high dependence on the quality of the mesh still demonstrates that accurate results can be obtained efficiently. The efficiency is demonstrated by comparison to the in-house 2D incompressible finite volume solver for flow around a circular cylinder. For this case the unstructured MAC method produced a solution four times faster than the finite volume code.

Declaration and Statements

DECLARATION

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed .... (candidate) Date ..24./04./2013..................

STATEMENT 1

This thesis is the result of my own investigations, except where otherwise stated. Where correction services have been used, the extent and nature of the correction is clearly marked in a footnote(s).

Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed .... (candidate) Date ..24./04./2013..................

STATEMENT 2

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ... . (candidate) Date ..24./04./2013..................

Contents

Acknowledgements

I would like to thank my supervisors Professor Oubay Hassan and Professor Ken Morgan for their continued advice and guidance and the EPSRC for the providing the doctoral training grant, without which this project would not be possible.

My friends Kevin Mason, Bruce Jones, Enayat Zangiabadi, Sean Walton and Hannah Buckland deserve thanks for providing valuable discussions whether related to my work, their work or other related fields. I especially thank Sean Walton for providing suitable meshes produced from his work on co–volume mesh optimisation. Finally I express my gratitute to my family and my husband Carwyn Pelley for their continued support and advise throughout the duration of the project.

List of Figures

List of Tables

Nomenclature

$\boldsymbol{\tau}$	Unit tangent vector to an element edge
$\boldsymbol{\tau}_o$	Anti-clockwise unit tangent
Δt	Time step size
μ	Viscosity
ψ	Pressure correction value
ρ	Density
τ_w	Wall shear stress
$\boldsymbol{n}_o$	Outward unit normal vector
$\boldsymbol{u}^*$	Dimensionless velocity vector
$\boldsymbol{x}$	Space vector
$\boldsymbol{x}^*$	Dimensionless space vector
$\tilde{\boldsymbol{u}}$	Intermediate velocity
$\tilde{p}$	Initial pressure
A_E, A_W	Areas of neighbouring Delaunay elements
A_V	Area of a Voronoï cell
D_E	Delaunay cell circumcentre which for a particular edge the normal points towards
D_W	Delaunay cell circumcentre which for a particular edge the normal points away from
E	Internal energy
e	General edge in the mesh
F_D	Drag Force

F_L	Lift Force
h^v	Voronoï edge length
h^d	Delaunay edge length
k	Thermal conductivity
L_0	Reference length
p	Pressure
p^*	Dimensionless pressure
P_E	Pressure location which edge normal points towards
P_W	Pressure location which edge normal points away from
Re	Reynolds number
s	Number of sides around an element
S_E	Energy source terms
S_m	Momentum source terms
T	Temperature
t	Time
t^*	Dimensionless time
t_0	Reference time
u_0	Reference velocity
u_1	Horizontal velocity component
u_2	Vertical velocity component
V_N	Voronoï cell centre tangent of an edge points towards
V_S	Voronoï cell centre tangent of an edge points away from
x	Cartesian x direction space vector component
y	Cartesian y direction space vector component
$\boldsymbol{n}$	Unit normal vector to an element edge
$\boldsymbol{u}$	Velocity vector

BLAS	Basic Linear Algebra Subroutines
C	CFL number
CFD	Computational Fluid Dynamics
CFL	Courant-Friedrichs-Lewy number
CG	Conjugate Gradient
m	Time step
MAC	Marker and cell
PCG	Preconditioned Conjugate Gradient
PDE	Partial differential equation
SMAC	Simplified Marker and cell
SOR	Successive over relaxation

Chapter 1

Introduction

The work described in this thesis aims to develop an efficient computational fluid dynamics (CFD) solver for the unsteady incompressible Navier-Stokes equations. The solver should be capable of simulating flow around arbitrary complex geometries, therefore making an unstructured grid approach attractive.

In the classical Cartesian mesh framework, the marker and cell (MAC) algorithm has been shown to be one of the most efficient techniques [7]. Therefore an investigation into applying the MAC algorithms techniques in an unstructured framework is carried out.

This chapter provides background on CFD and the MAC method, the objectives of this work will then be described, followed by a brief account of how the project developed over the course of the work. Finally the outline of this thesis is discussed.

1.1 Background

This section provides a brief introduction to the field of CFD, focussing on the scope and goals of this project. As the scope of the work is incompressible flow, this is then followed by a discussion on incompressible flow and the modelling techniques it requires. This section also details the history of the marker and cell method, including any previous attempts at incorporating an unstructured framework.

1.1.1 Computational Fluid Dynamics

Mathematically, a general fluid flow is governed by the Navier-Stokes equations [8, 9, 10, 11]. These equations are a set of non linear partial differential equations which consist of a momentum equation for each velocity component, the continuity equation, an energy equation to model heat transfer and an equation of state. Depending on the flow problem to be modelled, for example the flow may be compressible or incompressible, only some of these equations may be required.

An analytical solution is yet to be found for the Navier-Stokes equations except for a few specific problems. The solution to some problems can be obtained through experiment but then the necessary equipment to simulate the problem and record results are needed. Therefore, in order to gain a general solution potentially to all fluid flow problems, the field of computational fluid dynamics emerged. A good quality CFD solver can aid in cases where experiments are too costly and has the added benefit of producing solutions to problems that cannot be simulated experimentally.

CFD has applications in many areas, these include: the aerodynamics of aircraft and vehicles [9, 11]; environmental problems such as breaking dams or the spread of pollutants [11, 12]; the flow of blood through the body (a particular example includes the flow through arteries) [11]; injection moulding modelling [13, 9] and modelling of flow around ships [11, 14, 9]. These include just a few applications, many more can be found in [11, 9]. The role of CFD varies depending on the application, in engineering cases such as aircraft and vehicle aerodynamics it can act as a design tool. Many concept ideas can be modelled using the CFD software, the results of which can be analysed to determine the best design [9]. An alternative role may be to test an existing designs performance under varying flow conditions [9].

The basic ideas behind CFD is to convert the continuous Navier-Stokes equations to a discrete problem. A discrete problem is produced by dividing the problem domain into a series of sub-divisions, each variable in the Navier-Stokes equations can then be approximated at each of these sub-divisions. This allows the computation of a numerical solution. Although these computations can be done by hand, the scale of the problems in need of solving is so large, that this would be a very time consuming task. Therefore it is necessary that the approximations can be programmed allowing a computer to carry out all the computations. Historically computer power and memory meant that CFD simulations were restricted for use with problems that could be sub divided into a small number of points, which would then require a long run-time. However, this is no longer an issue due to the improvement in computer technology, numerical solutions are now obtainable for almost any geometry of any size [9]. The restriction on the run time still exists and the larger the size of the problem the longer the computation time. Therefore, one avenue of continual research in the CFD community is the reduction of the simulation time of general CFD problems. There are many works in the literature further describing the basic formulation of CFD, this work takes influence from the books by Versteeg et.al. [11], Anderson [9] and Ferziger et.al. [10]. The interested reader my refer to these items for more information.

In deriving a CFD solver, a number of choices must be made. The first of these is to decide on the form of the Navier-Stokes equations. The Navier-Stokes equations can either be represented in a Lagrangian or Eulerian frame of reference. A Lagrangian frame of reference can be understood as moving with the fluid whereas an Eulerian frame of reference can be seen as watching the fluid moving from a fixed point. For the purpose of this work we are concerned with the Eulerian frame of reference, thus it is that form of equations that is to be taken.

The next choice to make is that of compressible or incompressible flow. Not only do they model very different flow characteristics but they also require a very different approach to producing a numerical solution. The main cause of this difference in numerical solution is due to a change in the property of the density. In a compressible flow, density may vary over the fluid domain. A varying density effects mass conservation and so its effects are incorporated into the continuity equation. A desirable consequence of compressible flow is that an ideal gas can be assumed, therefore the ideal gas law can be used to find the pressure [9, 11]. For an incompressible flow, density is constant and so mass conservation is only determined by the fluids velocity. Another consequence is that pressure is no longer a thermodynamic variable and so the ideal gas law does not apply. Aside from density being constant, a simplification occurs with the energy equations, which may be discarded, as heat transfer does not take place in an incompressible fluid [9, 11].

After the correct form of the Navier-Stokes equations have been determined, the first thing needed to solve the Navier-Stokes equation is a mesh of the domain for the problem that is being simulated. A mesh gives the necessary sub-division of the problem domain in order to construct the discrete approximation. Each variable in the Navier-Stokes equations is approximated at locations defined by the mesh. A mesh of the domain links a set of nodes, that lie inside the domain by a set of lines, to form shaped elements. These elements are usually triangular or quadrilateral but this is not a restriction. A comment should also be made that for the purpose of this work all meshes are constructed by using straight lines to link the nodes.

A mesh can either be structured or unstructured. A structured mesh consisting of quadrilateral elements is based in a Cartesian coordinate system and is the historic approach to take [12, 15, 16]. Most introductory CFD texts describe this approach [9, 10, 11]. Structured meshes need not use quadrilateral elements, they may be triangular or in theory a shape with any number of sides. A structured mesh has a node layout that allows the identification of each node from neighbouring node using some known algorithm [17]. Elements within a structured mesh need not all be the same size, they may vary in size in block areas [16, 15, 9]. This allows the refinement of meshes in areas where more detail is required.

When using a structured Cartesian mesh framework, there is a difficulty in fitting a mesh around boundaries that do not follow the axis of the mesh. For cases where a curved boundary is required, special boundary conditions can be built into the solution algorithm to try and represent the true boundary of the domain [13]. An alternative approach (rectifying the problem) is to use a body-fitted mesh. Body-fitted meshes still fall into the structured class of meshes and require a transformation of the Navier-Stokes equations into general curvilinear coordinates. This approach requires extra computations to account for the transformations [18, 19, 20, 21].

The other more general option to fit meshes around any shaped boundary is to use an unstructured mesh. In an unstructured mesh, nodes are scattered over the problem domain and are linked as triangles and in some cases as quadrilaterals [17, 22, 23, 24]. An unstructured mesh offers a versatile approach and mesh elements can be constructed around any shape of domain or object within a domain. The density of the nodes can vary throughout the domain, allowing a finer resolution in any areas it may be required.

All unknown variables in the Navier-Stokes equations will be approximated at various points in the domain as defined by the domain mesh. There are various choices of variable location for a given mesh. The obvious location choice is to collocate all variables and locate them at either the nodes or the mesh element centres. Another possible choice is to store the variables at the mid points of the edges of an element [25, 11]. An alternative approach to collocating the variables is to stagger them over a mesh element, this avoids what is known as the chequer boarding pressure effect [15]. When using a collated method, compensating measures must be introduced into the equations approximations, this is not needed when using a staggered mesh. An example of a staggered mesh is described as follows, for an element in the mesh, the pressure may be located at the centre of the element, whereas the velocity can be located at the edge mid points [15, 12, 26]. When using a staggered mesh, the variable location must be taken into account within the equation approximations. This can sometimes result in a need for the interpolation for some variables.

The next stage in the production of a numerical solution for fluid flow is to define a set of discretisations which approximates the Navier-Stokes equations. These will be based on the type of mesh used, whether it be structured or unstructured, collocated or staggered. For an Eulerian formulation there are three main methods of discretisation: finite difference; finite volume and finite element. The finite difference case approximates derivatives by using the difference of neighbouring unknowns. It is usually used on structured Cartesian meshes, due to a dependence on knowing the locations of neighbouring nodes [9, 12, 10]. The finite volume case takes the integral form of the Navier-Stokes equation and integrates over control volumes (usually a cell in the mesh), to find the solution. The finite volume method can be used with Cartesian and triangular meshes [11, 10, 15]. The finite volume method is very closely related to finite difference methods and shares some of the same techniques, however it is very easily applied to meshes with any shaped elements including those of an unstructured nature. The finite element case uses the weak form of the Navier-Stokes equations coupled with basis functions to approximate a solution [27]. It may be used with both quadrilateral and triangular elements. Both the finite volume and finite element methods are most common for engineering problem types, as they offer more flexibility than finite difference methods due to their ability to use unstructured meshes.

The next required component is a solution algorithm. The solution algorithm is defined as the steps that need to be taken in order to find the solution to the Navier-Stokes equations. A discretisation of the Navier-Stokes equations is formed for a particular node, element centre or edge midpoint, depending on the choice of mesh layout. The goal is then to solve for the desired unknowns at all variable locations as defined in the domain mesh, a solution algorithm describes a method of achieving this. The choice of algorithm will vary depending on the type of discretisation and the type of flow to be modelled. For example, an incompressible flow may require the calculation of pressure to be decoupled from the velocity [12, 15, 28]. For all solvers the solution of some discrete equation defined for all unknown variable locations is required. This means there is a system of simultaneous equations to be solved in order to recover a full solution for the entire domain. These equations of course can be written as a matrix system, thus requiring the inversion of a matrix to solve. The discretisations may be solved explicitly or implicitly the choice of which will govern the form of the system matrix. When an explicit approach is used the matrix will be diagonal. However, if implicit time stepping is used off diagonal elements will appear in the matrix and so part of the solver will require the efficient inversion of this matrix. Both approaches have advantages and disadvantages and the choice greatly depends on the type of problem being modelled.

Regardless of the type of mesh and solver being used, it is advantageous that they are independent. This is beneficial in several ways: one being that it allows the re-use of a mesh, meaning CPU time is saved as meshes are not continually regenerated; another being the solver is not restricted to solving one particular problem, provided the mesh can be produced in a framework the solver can interpret.

As the work in this thesis is concerned with developing an efficient solver, something should be said about the CPU time required in a CFD simulation. The first constraint on the run time to mention is the mesh. As a mesh is refined, more nodes are introduced to the domain, this increases the number of unknowns to be found. Thus, increasing the size of the matrix system previously mentioned and requiring more CPU time. A number of techniques can be applied to reduce this increase in the degrees of freedom whilst retaining the fine resolution where it is needed. These include a mesh of varying element sizes, as the resolution may only need to be fine in certain areas. Further to this, high resolution may only be required in a certain direction, for example when modelling viscous flows, high resolution is only needed in the normal direction to a boundary. Here stretched elements can be used [29]. One other technique to mention, is that of using higher order methods. Higher order methods may use a coarser mesh resolution yet simulate a fine mesh by using higher order approximations in the discretisations [30]. Substantial run time savings can be made by employing any of these techniques.

The next constraint on the CPU time to mention is of the obvious one, the hardware. Of course, run a solver on a machine with a faster clock speed and a solution will be produced in a shorter amount of time. Further to this, recently there has been work into changing the type of hardware, utilising parallel implementations on GPUs instead of CPUs, offering substantial run time savings [31].

A particular solver will still have a run speed that will be slower than another and so regardless of the type of hardware being used different run times will still be observed when different solution algorithms are used. This leads on to the solution algorithm itself being a constraint on the CPU time. There are many choices for the solution algorithm and some do outperform others. In particular there is a class of old Cartesian algorithms for the solution of unsteady incompressible flow that have proved very efficient [12, 28]. Perhaps these methods can be converted for use with unstructured meshes to give a more efficient solver than those currently in use.

It should always be noted that CFD techniques are an approximation of a mathematical model, their solution is not exact. Steps can be taken to improve accuracy, such as using finer mesh resolutions or using a more accurate equation discretisation. Coupled with this, the Navier-Stokes equations are a model of the physical processes, therefore, solutions will differ from experimental. When chaotic random elements are included, like turbulence, accurate approximations are even harder to obtain. This means that any results of a CFD solver must be justified. To do this, solutions are compared to benchmark solutions which may be analytical solutions, experimental data or a historical numerical result. Only then can simulations to new problems be considered as having an accurate solution.

1.1.2 Incompressible Flow

In an incompressible flow, density is constant over the entire domain. In terms of the Navier-Stokes equations a constant density can be seen as a simplification. The continuity equation no longer changes over time with density. Instead, mass conservation is enforced through the velocity, its divergence must equal zero, i.e. over the domain the amount of fluid leaving will equal that entering. This applies to each element in a discretised domain and holds at all points in time. An exception to this are stratified flows, they may have a varying density through the fluid, however, the density remains the same at each point and so the incompressible Navier-Stokes equations are still applicable. This work is not concerned with the modelling of stratified flows and so a constant density throughout the domain is applied.

For a compressible flow, in most engineering problems an ideal gas can be assumed and so the ideal gas law can be used to link the pressure and density. This density evolves through time by the continuity equation and so a new pressure can be determined at each point in time [11]. As the ideal gas law does not apply to an incompressible flow, an equation for determining the pressure does not exist. This means there are now less independent equations than unknown variables to be found. The solution to this problem is to define a method to calculate the pressure, thus removing it as an unknown variable. In determining a numerical solution there a two commonly used methods to accomplish this task, these being the artificial compressibility method [32, 33, 34, 6] and the class of fractional step methods [12, 15, 11, 9, 28, 35].

Fractional step methods, also known as projection methods or pressure correction methods have many different forms. The general idea behind them is to split the velocity and pressure calculation. A common approach taken by many of the various methods is to solve the discrete momentum equations using a guessed value for the pressure. The velocities obtained in this calculation will be slightly incorrect, therefore a correction needs to be found. An equation for this correction can be formulated from the Navier-Stokes equations, where it is usually in the form of a Poisson equation. The velocity and pressure are then corrected via equations that are found as a consequence of the pressure correction equation. The process is then repeated with the corrected values being the new guess values. There are many types of these methods, examples of which include the SIMPLE method [15, 11, 9], the Marker and Cell method (MAC) [12, 36, 37] and the projection method [28, 35]. These methods were all originally developed for use with Cartesian meshes using finite difference techniques. The SIMPLE method however, was originally a finite volume solver for steady state flows [15]. This is achieved by solving the discrete equations using the solution algorithm over the entire domain. This is then done multiple times and at each time the obtained solution is fedback as the initial requirements for the next step. This time loop is repeated until the solution has suitably converged to the steady state. The time loop in this case is known as a pseudo time loop, it is not real time, just an iterative loop to progress the solution. The method can be adapted to solve for unsteady flows, via the means of introducing another iterative loop to advance in time. The marker and cell method however, allows for the direct solution of unsteady flows and so presents a very efficient scheme [12, 36].

The artificial compressibility method is a steady state incompressible flow solver originally devised by Chorin [32]. It introduces a time derivative for artificial density in the continuity equation. Coupled with this, an artificial equation of state linking the artificial density to pressure is defined. In this equation the artificial compressibility parameter is introduced. This parameter can be viewed as a relaxation parameter. The actual solution does not depend on its value but it has the ability to effect the rate of convergence and stability. The time derivative introduced is not real time and is often referred to pseudo time. It can be seen as being equivalent to the time used in compressible flow solvers, hence allowing solution by schemes developed for compressible flows to be used [33]. The artificial compressibility method has been extended to time accurate flow simulations using dual time stepping [6]. Dual time stepping was introduced by Jameson in 1991 [34]. It allows time accurate solutions by the use of an outer real time iterative loop and an inner pseudo time iterative loop. Within the pseudo time loop the fluid flow problem is solved to a steady state. This steady state solution is then used in the real time loop to add the residual fluxes to the previous time step solution. For the case of an actual steady state solution, these residuals would be zero. The artificial compressibility method is highly developed and can model flows using hybrid unstructured meshes [33].

The artificial compressibility method was originally developed as an efficient alternative to the fractional step methods for steady state flows. The fractional step method can be slow as it requires the solution of a Poisson equation. The discrete approximation to a Poisson equation is an implicit problem, which has the most costly overheads in a fractional step solution algorithm. However, in the artificial compressibility method when time accurate solutions are required, a solution to steady state is needed at every time step. It is therefore of interest to see if a time accurate fractional step method, like the marker and cell method, can produce a more efficient alternative in the unstructured mesh framework.

1.1.3 The MAC Method

The marker and cell method is a fractional step solver for the unsteady incompressible Navier-Stokes equations. It is of interest since in its Cartesian form, it has been shown to be one of the most efficient methods of its type [25, 7]. This makes it of interest for use as an efficient unstructured mesh solver. This section looks at the history of the method, its advancements, free surface capabilities and past attempts at using its ideas with an unstructured mesh.

The classical MAC algorithm is a computationally efficient co–volume solution technique for incompressible flow problems with a free surface, using a staggered Cartesian mesh [12, 16]. A co–volume solution technique requires a dual mesh with the centres of the primary mesh being the vertices of the dual mesh. These vertices are used to define the location of the pressure variable. The locations of the velocity variables are defined at the intersect of the primary and dual mesh cell edges, in the Cartesian framework a further staggering takes place and the velocity components are split. This means that only the normal component is located at a particular edge. In order for the MAC scheme to remain stable the primary mesh and its dual are required to be dual orthogonal, meaning that element edges are perpendicular bisectors of each other. Of course in a Cartesian framework this presents no issue when square elements are defined and it is in this framework that the method first appeared using a pair of staggered square meshes.

The method was developed primarily by Harlow and Welch in the 1960s and as an improvement of the particle in cell method (PIC) method as discussed in [38]. The major further developments in the method using a Cartesian grid framework were carried out by the same research group as the original 1960s work [12, 37]. The main objective of the MAC was the simulation of free surface flows, although it is easily possible to omit this capability leaving just an unsteady incompressible flow solver.

The original marker and cell method overcame the problem with finding the pressure by manipulating the Navier-Stokes equations. The momentum equations are differentiated with respect to the direction they represent. In two dimensions the horizontal velocity is differentiated with respect to the x direction and the vertical with respect the y direction. The resulting equations are then added together obtaining a Poisson equation for the pressure, the right hand side of which is known and only depends on the velocities [39, 16]. The solution algorithm begins by solving the discrete version of the pressure equation using either initial velocities or those from the previous time step. The discrete versions of the momentum equations are then solved to obtain the velocity vector. The process is repeated for each time step.

The method has been improved greatly since its first creation. Calculation of the pressure is a little on the complex side and so the first improvement came very soon after its first publication. This altered the MAC method to the simplified marker and cell method (SMAC) [37]. Not only did this improve the efficiency of the scheme but it also altered the way in which the pressure is calculated. Instead of finding an equation for the actual pressure, a Poisson equation for a pressure correction is used and the pressure and velocity are corrected using this value. The formulation of the pressure correction equation presents itself in a much simpler form than the original pressure equation, the right hand side of the Poisson equation is now just equal to the divergence of the velocity. The sequence of the solution algorithm was then altered slightly to incorporate this technique. The initial stage for SMAC solves the momentum equations using a guess pressure, the pressure correction values are then found, following this step the velocity and pressure values are then corrected. The SMAC algorithm will be the basis of this thesis and so a more in-depth account of the stages it involves will be given later.

The original MAC and SMAC algorithms are explicit techniques and further alterations to the methods implemented implicit time stepping schemes. Using an implicit time stepping scheme removed the time step dependence on the viscous term, therefore allowing the modelling of very slow flows [40, 39]. A semi-implicit scheme SIMAC has also been developed for the purpose of modelling high Reynolds number flows on stretched meshes [41], in which only the viscous term in the Navier-Stokes equation is solved implicitly. A notable recent set of developments are those by the group developing the GENSMAC algorithm [13]. GENSMAC is a SMAC based algorithm tailored for modelling injection moulding problems, therefore incorporating free surface simulation into the method [13]. The algorithm still uses a Cartesian mesh but achieves successful modelling of the complex geometries required in injection moulding through the use of specially defined boundary conditions. GENSMAC was originally devised as a 2D Newtonian flow solver [13], with later developments to Non-Newtonian fluids [36]. Later the solver was extended to three dimensions in GENSMAC3D [42, 43]. Reviews on the use of the MAC method in the GENSMAC context can be found in works by Mckee et.al. [44, 38]. Most recently, development of a semi-implicit GENSMAC scheme was implemented, with a comparison of the various time stepping schemes to assess the stability [45, 46, 47]. The semi-implicit schemes only treat the viscous term implicitly as with SIMAC. This is due to the viscous term producing the most restrictions on the time step size. Therefore, for successful modelling of slow flows, or flows that require stretched meshes, the semi-implicit approach allows a necessary increase in the minimum time step.

Injection moulding fully utilises the marker and cell methods free surface capabilities. This is accomplished by defining massless particles which move with the fluid. Mesh elements which contain these marker particles are classed as an area containing fluid or a surface area. The free surface modelling aspect of the marker and cell algorithm is another area which has been developed. Initially, the pressure on the free surface was not calculated and pressure was set to zero on cells outside of the fluid. This approximation was improved in 1971 to incorporate the calculation of normal and tangential stress on the free surface [48]. For certain types of problem, for example that of breaking waves, this free surface stress condition was not adequate and so different stress conditions were applied in both two and three dimensions to simulate this phenomena [14, 49].

The development having the most impact on the free surface modelling community is the Volume of Fluid method (VOF). This was first shown in the SOLA-VOF code as an advancement of the MAC approach [50]. VOF no longer required the storage of many marker particles per mesh element, instead a value of one or zero is stored for each element depending if an fluid is contained in that element. Although the VOF method reduced storage costs, it increased difficultly of locating the exact position of the free surface. This aspect of the method has been highly developed as well as coupled with other free surface techniques, such as the level set method, in an attempt to use the VOF methods capability of conserving mass and still retain accurate interface capturing [51, 52, 53, 54]. As mentioned, the VOF method has been well developed and has been extended to use with unstructured meshes in two and three dimensions [55, 56]. The MAC approach, however, has not been abandoned and it could be argued that for certain applications it still provides the best approach, hence its use by the GENSMAC group for injection moulding.

The discretisations of the Navier-Stokes equations on a domain represented using the MAC staggered mesh framework, has been extended by several authors. The initial extension appeared at around the same time in work by Nicoliades and Hall, a collaborated review of their work can be found in [57]. The work by Nicoliadies has been mainly on the discretisation of div-curl problems on triangular meshes, which has been easily extended to three dimensional tetrahedral meshes [58, 59]. Later work developed a fourth order accurate scheme, with the order being proved on structured meshes [60]. Further work in the same group uses the discretisations for vorticity-velocity functions [61]. The work by Nicoliadies, focus' more on the mathematical side rather than application of the method. The mathematical analysis of the discretisations for the Navier-Stokes equations is carried out and the order of accuracy for the velocity is proved to be second order, also stated is the belief that the pressure is first order accurate [62, 63]. No results for numerical solutions using the Navier-Stokes equations are reported in their works. Further to the work by Nicoliades is the work by Hall et.al. the same discretisations are also presented for the Navier-Stokes equation, however, in the case of Hall et.al. they are solved using the Dual Variable Method [64]. Successful results for CFD problems such as the lid driven cavity and flow around a circular cylinder are presented. Further to this work are three dimensional implementations and an alternative discretisation for the convective term which uses a directional derivative [65, 66]. The alternative convective term allows for an upwind difference scheme to be used, although, this has lower accuracy.

Further attempts to extend the marker and cell methods ideas have been carried out by several authors, the most notable being by Perot [67], Vidovic [68] and Wenneker [69]. Perot again made use of the marker and cell staggered grid system and similar discretisations appear, however, both the divergence and curl form of the Navier-Stokes equations are considered. Once again the MAC approach is not used to solve for the pressure. This scheme has been extended to three dimensions and for use with moving meshes [67, 70, 71]. Other uses of the marker and cell methods ideas have been the coupling of the staggered mesh with an artificial compressibility method with some using the staggered mesh for compressible flow simulations [68, 69]. As of yet, the attempt to couple an unstructured staggered mesh with the marker and cell solution algorithm has not been found in the literature.

1.2 Objectives

The main objective of this work is the development of an efficient Eulerian solver for the unsteady incompressible Navier-Stokes equations. The solver needs to be capable of modelling flows around complex geometries. For reasons demonstrated in the previous section an old Cartesian mesh solver named the MAC method has been chosen as the basis for this efficient solver. The initial objective of this work is to convert the two dimensional Cartesian MAC method for use with an unstructured triangular method. The staggered mesh layout is desirable to keep as it eliminates problems with the checker boarding pressure effect [15]. Therefore, within the solver a staggered mesh layout for use with triangular meshes needs to be selected and suitable discretisations obtained. In fact, the MAC staggered mesh system naturally allows using the Delaunay-Voronöi diagram, a Delaunay triangulation is commonly used as a mesh generation tool [17].

The next objective is to test the two dimensional scheme against a variety of steady and unsteady state benchmark problems using a variety of meshes, addressing short comings in the method at each stage. Since the MAC method is a co–volume scheme it depends on the use of dual orthogonal meshes, under testing the limitations of this constraint should become apparent.

Originally it was conceived that the two dimensional case could be extended to three dimensions, however, after testing and the extra time needed to overcome the discovered problems with the high dependence on suitable meshes, this objective was dropped. Instead, enabling the solver to model viscous flows using hybrid stretched meshes replaced this objective.

The next section details the development of the project over the time period, giving information of the problems encountered and the particular avenues of research that were taken. Some works were dropped and so are not presented in this thesis, however, the following section presents the opportunity to mention all that was attempted.

1.3 Project Development

The initial stage of this project began with the reading of literature detailing the basics of CFD and the MAC algorithm. As a learning exercise the structured mesh algorithms for the MAC and SIMPLE algorithms were implemented for the simple test case of the lid driven cavity [1] and were compared.

After this initial stage, research began on the possible implementation of an unstructured MAC algorithm. This led to the discovery of the discretisations of Hall et.al [64], which used an unstructured MAC mesh. The unstructured formulation of the MAC algorithm was devised shortly after this.

Work could then begin on implementing the unstructured MAC solver into a Fortran 90 code. The initial basis of this work was the in-house source code for the Yee algorithm [24], a co-volume technique used in electro-magnetics. Provided in the Yee algorithm were the basic algorithms for manipulating an unstructured mesh. The decision has been made to implement the MAC code in Fortran 90 to take advantage of the dynamic data structures, this meant the re-used elements of the Yee algorithm needed to be converted from Fortran 77.

The first successful results using the unstructured MAC algorithm were validated using the lid driven cavity test case. That was shortly followed by modelling flow around a circular cylinder, which allowed the implementation of suitable inflow and outflow boundary conditions. The flow around a circular cylinder presented a diverse set of problems, as well as being the first unsteady simulation it also has an exact solution when inviscid flow is being simulated. These boundary conditions were later backed-up using the flow over a backward facing step test case.

Further testing of the MAC solver using aerofoil geometries flagged up a variety of problems. These included discovering the algorithms high dependence on dual orthogonal meshes. Solutions became difficult to obtain when the mesh deviated from the dual orthogonal properties. A problem that was later solved by the mesh optimisation algorithm developed by Walton et.al. [72]. The aerofoil test cases saw a drop in performance of the algorithm which was later attributed to the conjugate gradient algorithm, the convergence of which slowed when solving using large meshes with huge variants in the mesh element sizes. The decision was therefore taken to implement a direct solver for the solution of the pressure correction equation.

Using aerofoils with high Reynolds number flows led to the use of hybrid meshes (meshes which combine both triangular and quadrilateral elements), which again caused a range of problems with the MAC algorithm. The first stage was to convert the code so a hybrid mesh file could be interpreted. Following this, stable solutions were difficult to obtain with the hybrid meshes, this was attributed to a loss in orthogonality, particularly around tail regions of the aerofoil. Further demonstrating the need for these dual orthogonal meshes.

A brief investigation into pre-conditioners for the iterative solver was implemented and testing with skewed meshes for the lid driven cavity problem. This work was dropped due to the use of a diagonal preconditioner and ILU(0) preconditioners showing no improvement in speed for the test cases implemented.

During this time an investigation into the use of the unstructured MAC algorithm as a free surface problem solver was carried out. Initial results were promising when marker particles were used to track the surface. Further research was carried out in an attempt to make this process up to date, which led to a crude version of the VOF method being implemented. The surface capturing element of the VOF method on unstructured mesh posing a real problem.

During the development of the free surface work, a direct solver had been implemented into the algorithm and a huge decrease in the CPU time was observed. It was decided that concentrating on the codes capabilities at modelling viscous flows would make the most of the efficiency of the algorithm. Therefore, the free surface implementation was stopped and the investigation into using hybrid meshes with highly stretched boundary elements recommenced. It was noticed that the time step size would become incredibly small when using these stretched meshes and so a semi implicit scheme was implemented to overcome the problem.

During the project two papers were presented at conferences, the first in March 2010 at the ACME2010 conference held at Southampton University [73], the second in March 2011 at the FEF2011 conference held in Munich [74]. A journal paper detailing the explicit scheme is also in development, with the potential for a second on the implicit scheme.

The following list gives a summary of the main stages taken within the project:

1. Initial research into the history of the MAC algorithm.

2. Discovery of a discretisation for the unstructured MAC staggered mesh.

3. Adaptation of the MAC algorithm for use with unstructured meshes.

4. Formulation of discretisations for the pressure correction equation.

5. Development of the unstructured MAC code using the conjugate gradient method to solve the pressure correction equation.

6. Testing of the algorithm with benchmark problems. This allowed refinement of the boundary condition implementation.

7. Testing of the algorithm for flow around aerofoils. This identified problems with meshes that contain elements with cell circumcentres outside the triangles. When meshes were used which did not have this property stable solution were obtained.

8. A direct solver was then implemented to solve the pressure correction equation. This allowed a dramatic speed decrease demonstrating that most computational effort is in the solution of the pressure correction equation.

9. Meshes with stretched boundary elements were then tested. In doing so difficulties occurred in obtaining vortex shedding in any reasonable amount of run-time. This was attributed to the reduction in time step size caused by the stretched elements.

10. An semi-implicit method was then implemented to over come the time step size problem.

1.4 Thesis Structure

This thesis details all the necessary information to reconstruct the MAC solver, therefore all its theory and code implementation are detailed. Background information directly relating the work has been supplied but is not intended to provide an exhaustive overview of the entire field of CFD. The field of CFD is incredibly large, a lot of which, is not needed for the production of this work. Issues with the development of the method are also detailed, providing justifications for features that have been added to the solver. Of course, the use of the unstructured MAC method needs to be justified and a chapter is dedicated to displaying results for various fluid flow problems. Conclusions are then drawn on the results and presented in the final chapter. A brief overview of the contents of each chapter will now be given.

Chapter 2: In chapter 2 the governing equations will be introduced. A solver has been developed for a two dimensional flow and so it is the two dimensional Navier-Stokes equations that are given.

Chapter 3: Chapter 3 describes the unstructured MAC algorithm. In order to fully define the method this will include the mesh layout, the discretisations and the unstructured MAC algorithm. There are a number of other techniques required to fully implement the algorithm, all of which will be described in this chapter.

Chapter 4: In chapter 4, a discussion on mesh generation is given. The algorithm requires high quality dual orthogonal unstructured mesh and so the main algorithms in producing unstructured meshes will be discussed along with processes required for ensuring dual orthogonality.

Chapter 5: Chapter 5 provides a description on how a modular code has been implemented. Detailed are the input and output process and all data structures that were required.

Chapter 6: Chapter 6 gives numerical results. These include various benchmark problems to validate the code as well as problems which push the algorithm capabilities.

Chapter 7: Chapter 7 draws conclusions and discusses any future work that could be implemented.

Chapter 2

Governing Equations

The governing equations for fluid flow problems are the Navier-Stokes equations. For completeness the equations will be given for both compressible flow and incompressible flows. For the purpose of modelling using an unstructured MAC method, several manipulations are required, one being the conversion to dimensionless form. Other manipulations look at a form for the tangential and normal velocities and the curl and divergence form. All of theses along with the required boundary conditions will be detailed in this chapter.

2.1 The Navier-Stokes Equations for compressible flows

For compressible flows the Navier-Stokes equations include equations that fulfil Newtons second law, the first law of thermodynamics and mass conservation. To fully define all independent variables, an equation of state linking pressure to density and temperature is also required. Newton's second law is fulfilled by the momentum equation. The first law of thermodynamics is satisfied by the energy equation and to conserve mass, the continuity equation is defined. These governing equations are defined as,

$$\begin{aligned}
&\text{Momentum:} && \frac{\partial \rho \boldsymbol{u}}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}\boldsymbol{u}) = -\nabla p + \nabla \cdot (\mu \nabla \boldsymbol{u}) + S_m \\
&\text{Continuity:} && \frac{\partial \rho}{\partial t} + \nabla(\rho \boldsymbol{u}) = 0 \\
&\text{Energy:} && \frac{\partial \rho E}{\partial t} + \nabla \cdot (\rho E)\boldsymbol{u} = -p\nabla \boldsymbol{u} + \nabla \cdot (k \nabla T) + S_E \\
&\text{Equations of State:} && p = p(\rho, T) \qquad \text{and} \qquad E = E(\rho, T)
\end{aligned}$$

where $\boldsymbol{u} = (u_1, u_2)^T$ is the velocity vector in a Cartesian coordinate system and so $\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y})^T$. The pressure is denoted by p, t is time, μ is the viscosity, ρ is the density, E is the internal energy, T the temperature, k is the thermal conductivity of the fluid, S_m represents any momentum source terms and S_E are any energy source terms.

For clarity it is useful to refer to each term in the momentum equation by name,

$$\underbrace{\frac{\partial \rho \boldsymbol{u}}{\partial t}}_{\text{Temporal Term}} + \underbrace{\nabla \cdot (\rho \boldsymbol{u}\boldsymbol{u})}_{\text{Convective Term}} = \underbrace{-\nabla p}_{\text{Pressure Term}} + \underbrace{\nabla \cdot (\mu \nabla \boldsymbol{u})}_{\text{Viscous Term}} + S_m.$$

The temporal term allows for the evolution of a velocity component through time. The convective term adds the effects of a velocity component moving through the fluid. The pressure term allows for any pressure gradients to have influence on the fluids velocity. The final term, the viscous term, adds any resistance effects due to viscosity. In the convective term, $\boldsymbol{uu}$ represents the dyadic product.

2.2 The Unsteady Incompressible Navier-Stokes equations

The scope of this thesis is unsteady incompressible flow problems. Incompressible flow greatly simplifies the compressible Navier-Stokes equations due to the density being constant. The unsteady incompressible Navier-Stokes equations are composed of two parts, the momentum equations which describe the evolution of the velocity and the continuity equation. Pressure is no longer a thermodynamic variable and so it cannot be linked to density and temperature. Therefore, the equation of state and the energy equation can be neglected. Another property of incompressible flow is that viscosity is constant over the domain thus simplifying the viscous term. As long as the flow is not stratified, density is also constant over the entire domain.

The Navier-Stokes equations for unsteady incompressible flow are,

$$\text{Momentum:} \qquad \rho \frac{\partial \boldsymbol{u}}{\partial t} + \rho \nabla \cdot (\boldsymbol{uu}) = -\nabla p + \mu \nabla^2 (\boldsymbol{u}) \tag{2.1}$$

$$\text{Continuity:} \qquad \nabla \cdot \boldsymbol{u} = 0 \tag{2.2}$$

For an incompressible fluid, density is constant in space and time therefore the only variable affecting mass conservation is velocity. The amount of fluid entering a domain must equal that leaving.

2.3 Initial Conditions and Boundary Conditions

The concept of initial and boundary conditions is applicable for both compressible and incompressible flow, however, since the scope of this work is only incompressible flow, initial and boundary conditions relating to compressible flows are not detailed.

The Navier-Stokes equations are defined over a domain for the particular fluid flow problem. To fully define a fluid flow problem, initial and boundary conditions are required. For an unsteady incompressible flow, u and p must be defined at time $t = 0$ everywhere in the problem domain [9, 11]. The boundary conditions for an incompressible flow vary depending on the type of problem being modelled. There are several types of boundary conditions that can be applied, these are:

Wall: Fluid cannot flow through a wall and friction effects mean a fluids velocity is reduced to zero along it. Therefore, both velocity components are defined as zero on a wall boundary.

$$u_1 = u_2 = 0$$

Moving wall: For a moving wall the velocities on the boundary are set to simulate the velocity of the moving wall. This can be determined using many combinations depending on the problem to be specified. To illustrate the application of the moving wall boundary condition consider a problem where a wall is aligned with the x axis, this wall moves in the direction on the axis. To simulate this set up the normal velocity to the boundary is set to zero, whereas, the tangential component is set to the speed of the wall.

Inviscid wall: An inviscid wall has no friction effects and so the velocity component tangential to the boundary is not defined. Fluid still cannot flow through the boundary and so the velocity component normal to it is set to zero.

Inflow: The standard procedure on an inflow boundary is to set both velocity components to the required speed of the fluid entering the domain.

Outflow: Anywhere that fluid leaves the domain a Neumann boundary condition can be applied to all variables, i.e the gradient of the variable in the outward normal direction is zero. This has the effect of defining the fluid just outside the domain to have the same properties as that inside. A technique commonly applied in incompressible flow is to calculate the unknown velocities by manipulating the Navier-Stokes equations. The pressure is also unknown but its gradient in the outward normal direction can be defined as zero. Provided the outflow boundary is located far enough away from major fluid activity, this approach provides an adequate approximation.

This list is not exhaustive, in particular for an incompressible flow, other boundary conditions such as constant pressure, the symmetry condition and a periodic boundary condition can be implemented. These are not required in the scope of this work and so there representation has not been discussed. More information can be found on boundary conditions in the literature, in particular the works by Anderson [9], Versteeg et.al. [11], Sørensen [75] and Zhang [76] have been used to influence the descriptions detailed here.

2.4 Dimensionless Form

For the purpose of fluid simulation it is useful to have the equations in dimensionless form. This allows easy comparison of fluid flow problems on different scales. To obtain the dimensionless form each variable is normalised by a reference value. These are denoted as follows, reference velocity magnitude as u_0, the spatial coordinates have a reference length of L_0, a reference time t_0 and the pressure is normalised by ρu_0^2. The dimensionless variables are denoted by the superscript * and are defined as,

$$\boldsymbol{u}^* = \frac{\boldsymbol{u}}{u_0}; \qquad \boldsymbol{x}^* = \frac{\boldsymbol{x}}{L_0}; \qquad t^* = \frac{t}{t_0}; \qquad p^* = \frac{p}{\rho u_0^2}$$

where $\boldsymbol{x} = (x, y)^T$ is the space dimensions vector. Using the dimensionless variables the Navier-Stokes equations can be rearranged to,

$$\text{Dimensionless Momentum:} \qquad \frac{\partial \boldsymbol{u}^*}{\partial t^*} + \nabla \cdot (\boldsymbol{u}^* \boldsymbol{u}^*) = -\nabla p^* + \frac{1}{Re}\nabla^2 \boldsymbol{u}^* \tag{2.3}$$

$$\text{Dimensionless Continuity:} \qquad \nabla \cdot \boldsymbol{u}^* = 0 \tag{2.4}$$

A by-product of conversion to dimensionless form is the creation of a dimensionless number Re, known as the Reynolds number.

$$Re = \frac{u_0 L_0 \rho}{\mu}.$$

The Reynolds number is a measure of the ratio between inertial forces and viscous forces. A fluid flow problem in a particular geometry will display the same characteristics for a given Reynolds number. Two fluids could have a very different viscosity yet the traits of the flow will be the same, provided the reference velocity and length are altered so that the Reynolds number is equal. This meant large scale experiments could be made smaller and realistic results still be obtained. It is also useful in numerical flow simulations as the scaling properties allow the re-use of meshes for comparison with experimental results of different scales.

It should be noted that for convenience, the * superscript will now be dropped and the dimensionless variables will just be denoted by their letter.

2.5 Curl-Form of the Viscous Term

There are several forms of the Navier-Stokes equations that can be used. In equation (2.3), all terms are given in divergence form. This is easily identified by the use of the div operator in the viscous and convective terms. It is possible to write the equations in what is known as the curl form which is identified by the use of the curl operator in the viscous and convective terms.

For the purpose of this research it is beneficial to use the curl form of the viscous term and the divergence form of the convective term. The reasons for this are due to the ease in obtaining suitable discretisations which will become clear later on. Here however, the transformation of the viscous term from the divergence form to the curl form is explained. The transformation makes use of the vector identity, [64]

$$\nabla^2 \boldsymbol{u} = \nabla(\nabla \cdot \boldsymbol{u}) - \nabla \times (\nabla \times \boldsymbol{u}). \tag{2.5}$$

The identity equation (2.5) can be simplified by applying continuity, equation (2.4), the first term on the right hand side thus becoming zero. This means that the viscous term can be directly replaced by

$$-\frac{1}{Re}\nabla \times (\nabla \times \boldsymbol{u}),$$

giving the full momentum equations as,

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot (\boldsymbol{u}\boldsymbol{u}) = -\nabla p - \frac{1}{Re}\nabla \times (\nabla \times \boldsymbol{u}). \tag{2.6}$$

2.6 Momentum equation for the normal and tangential velocity component

The above form of the Navier-Stokes equations form multiple equations for all components of velocity in a Cartesian coordinated system. It is also useful to have them in a form which only solves for a normal or tangential velocity component. This is because in an unstructured mesh system the coordinate system for the velocities becomes local to particular edges of elements. To obtain the equations in the normal and tangential form, a right-handed coordinate system $(\boldsymbol{n}, \boldsymbol{\tau})$ is defined, where $\boldsymbol{n}$ is the unit normal vector and $\boldsymbol{\tau}$ the unit tangent. The direction derivatives can be reinterpreted as derivatives in the normal and tangential directions and so, $\nabla = (\partial/\partial \boldsymbol{n}, \partial/\partial \boldsymbol{\tau})^T$. The inner product of equation (2.6) with unit normal vector $\boldsymbol{r}$, gives a scalar equation for the normal velocity component in the new right-handed coordinate system,

$$\frac{\partial \boldsymbol{u}}{\partial t} \cdot \boldsymbol{n} + \nabla \cdot (\boldsymbol{u} \cdot \boldsymbol{n})\boldsymbol{u} = -\frac{\partial p}{\partial \boldsymbol{n}} - \frac{1}{Re}\frac{\partial}{\partial \boldsymbol{\tau}}(\nabla \times \boldsymbol{u}). \tag{2.7}$$

Since $\nabla \times$ can be written as $(\partial/\partial \boldsymbol{\tau}, -\partial/\partial \boldsymbol{n})^T$. The convective term has also been rearranged into an equivalent more convenient form. This is possible since the dyadic product is equivalent to $(\nabla \cdot \boldsymbol{u})\boldsymbol{u} \cdot \boldsymbol{n}$, the dot product is commutable and so the given form is obtainable.

Similarly the inner product can be taken with the unit tangent vector, obtaining the scalar equation for the tangential velocity,

$$\frac{\partial \boldsymbol{u}}{\partial t} \cdot \boldsymbol{\tau} + \nabla \cdot (\boldsymbol{u} \cdot \boldsymbol{\tau})\boldsymbol{u} = -\frac{\partial p}{\partial \boldsymbol{\tau}} + \frac{1}{Re}\frac{\partial}{\partial \boldsymbol{n}}(\nabla \times \boldsymbol{u}). \tag{2.8}$$

The usefulness of the equations in this form will become apparent when the mesh layout and discretisations are introduced. The normal and tangential velocity components can then be defined as u and v respectively, where $u = \boldsymbol{u} \cdot \boldsymbol{n}$ and $v = \boldsymbol{u} \cdot \boldsymbol{\tau}$

Chapter 3

A Two Dimensional Unstructured Marker and Cell Method

The theoretical components in producing a MAC method suitable for use with unstructured two dimensional hybrid meshes will now be detailed. This will include: the MAC staggered mesh layout in a triangular framework; the unstructured MAC algorithm; the equation discretisation in both explicit and semi-implicit frameworks; the boundary conditions; and discrestiation of the pressure correction equation. At the end of the chapter solution techniques for implicit systems are discussed.

3.1 Mesh Layout

To successfully implement the MAC method, a dual orthogonal mesh is required. Creating a dual mesh in a Cartesian framework is straight forward and consists of a quadrilateral primary mesh with its dual mesh constructed by connecting the centres of the quadrilaterals. This effectively forms a quadrilateral staggered mesh. In the Cartesian case only the normal velocity component to a particular element edge will be solved for. This means that either the horizontal or vertical component of velocity will be solved for, depending on the orientation of the edge. The velocity component is stored at an edge midpoint, which can also be interpreted as the intersection of the staggered and dual mesh edges. The pressure is stored at the cell centres which coincides with the dual mesh vertices, figure (3.1) gives an example of a Cartesian dual mesh and its variable layout. The staggering of variables allows for the momentum equations to be solved easily and with no adverse effects for the pressure. The particular staggered mesh layout defined in the marker and cell case is beneficial, as the velocities are located on

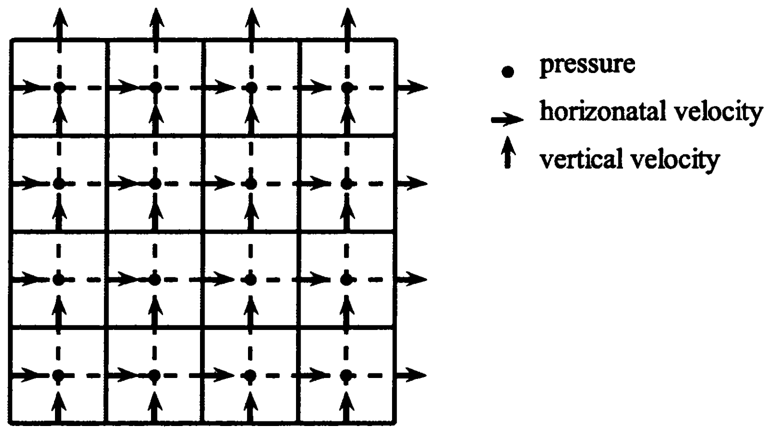


Figure 3.1: Cartesian dual mesh with staggered variable layout

the boundaries whenever the value is known, depending on the boundary condition, the velocity value can be set to the required amount without the need for any interpolation to a point within the domain.

A similar procedure to that of constructing a Cartesian dual mesh can be followed when obtaining an unstructured dual mesh. Unstructured meshes are constructed using many techniques, some of which will be discussed later. For the case of constructing an unstructured dual mesh, it is convenient to use a Dèlauany triangulation as the unstructured primary mesh. The circumcentres of the Delaunay triangles can be joined to form the dual mesh, this happens to be a Voronoï tessellation. An ideal mesh, that fulfils the dual orthogonality requirement a Delaunay triangulation of equilateral triangles. Using the ideal mesh, will allow second order accuracy for the velocity in the proposed scheme [60].

Although highest accuracy of the scheme requires an ideal mesh, the theory detailed is not dependent on that being the case and in theory it can be applied to any mesh. This is necessary, as when modelling flows around complex geometries, the use of the ideal mesh over the entire domain would not fit well to boundaries. In reality there are difficulties in using the unstructured MAC method when the mesh deviates significantly from the ideal case. This is not within the scope of this chapter, but the subject of generating suitable meshes for use in the unstructured MAC method and the requirements placed on that mesh will be discussed in detail in the next chapter.

The staggered arrangement of the variables in the unstructured case has a further difference from the Cartesian case. The velocity vector is no longer staggered and the full velocity vector is stored at cell edges. Pressure remains stored at element centres, which in the unstructured case are the Delaunay triangle circumcentres. Figure (3.2) gives an example of an unstructured dual mesh and its staggered variable layout. The

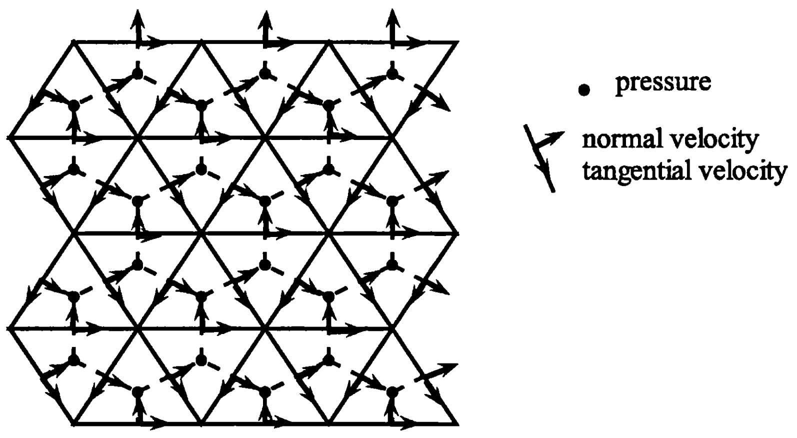


Figure 3.2: Delaunay-Voronoï dual mesh with staggered variable layout

velocity is now collocated due to the edges of cells no longer running parallel to the x or y axis, instead they have an arbitrary direction. A consequence of having edges of arbitrary direction makes it convenient to use a velocity vector that consists of normal and tangential velocity components that are local to a specific edge. Using normal and tangential velocity components means the standard form of the Navier-Stokes equations cannot be used. It is here that the need to have the momentum equation in terms of the normal and tangential velocity components, equations (2.7) and (2.8), becomes apparent. The solution is now required for the normal and tangential velocity components not the horizontal and vertical velocity components. For the devised unstructured MAC algorithm, the discretisations detailed by Hall.et.al [64] are used and only the normal velocity component to an edge is solved for. The normal velocity is the simplest form to discretisatise when using the MAC staggered mesh layout as given in figure (3.2). This staggered mesh layout makes solving for the tangential velocity component difficult, this difficulty can be seen in the pressure term in equation (2.8). The derivative of the pressure in the tangential direction would need a value for the pressure at the primary mesh vertices. Using the current mesh layout that would require some interpolation. After finding the normal velocity component the tangential velocity component still needs to be calculated and this is achieved by reconstructing it from the normal velocity component, another technique described by Hall et.al. [64].

It is possible to use a different staggered mesh layout and be able to use equation (2.8), the pressure could be moved to the primary mesh vertices, producing a staggered mesh layout similar to that of the SIMPLE algorithm [15]. In discretising the equation for the normal velocity, the pressure is required at the dual mesh vertices and a similar problem would occur in its calculation, to that of calculating the tangential velocity using the previous mesh layout. Therefore, the normal velocity would have to be recovered from the tangential velocity, making no benefit to using an alternative layout.

3.2 The Unstructured MAC Algorithm Derivation

To produce a time accurate solution to the incompressible Navier-Stokes equations, a valid time stepping algorithm is required. For unstructured hybrid meshes an implementation of the MAC algorithm is adapted from the cartesian GENSMAC algorithm [13].

The algorithm as given by Tome et.al. [13], begins by taking an initial pressure $\tilde{p}$ and an initial velocity $\boldsymbol{u}$. These values can then be used to solve the momentum equation for the normal velocity, equation (2.7). The specified initial pressure may not be correct and so the velocity obtained when solving equation (2.7) will be approximate. This approximate velocity is known as the intermediate velocity and is denoted by $\tilde{\boldsymbol{u}}$. Using the initial and intermediate values, the momentum equation for the normal velocity can be rewritten as,

$$\frac{\partial \tilde{\boldsymbol{u}}}{\partial t} \cdot \boldsymbol{n} + \nabla(\boldsymbol{n} \cdot \boldsymbol{u})\boldsymbol{u} = -\frac{\partial \tilde{p}}{\partial \boldsymbol{n}} - \frac{1}{Re}\frac{\partial}{\partial \tau}(\nabla \times \boldsymbol{u}). \tag{3.1}$$

Equation (3.1) is known as the approximate equation. The intermediate velocity produced from the approximate equation and the specified initial pressure must then be corrected. The correction is done by finding a correction value from the solution of a Poisson equation. The Poisson equation, known as the pressure correction equation, is obtained by manipulating the momentum equation for the actual velocity and the equation for the approximate velocity. The first step in formulating the pressure correction is to subtract the approximate equation (3.1) from the exact momentum equations equation (2.7). Giving the following,

$$\frac{\partial}{\partial t}(\boldsymbol{u} - \tilde{\boldsymbol{u}}) \cdot \boldsymbol{n} = -\frac{\partial}{\partial \boldsymbol{n}}(p - \tilde{p}). \tag{3.2}$$

The curl of equation (3.2) can be taken to obtain,

$$\nabla \times (\frac{\partial}{\partial t}(\boldsymbol{u} - \tilde{\boldsymbol{u}}) \cdot \boldsymbol{n}) = -\nabla \times (\frac{\partial}{\partial \boldsymbol{n}}(p - \tilde{p})). \tag{3.3}$$

The pressure term on the right hand side of equation (3.3) can be written as,

$$\nabla \times (\nabla (p - \tilde{p}) \cdot \boldsymbol{n}) \tag{3.4}$$

Equation (3.4) is the form before the simplication from taking the dot product with the unit normal has been made. Then using the fact that the curl of a gradient is equal to zero, ($\nabla \times \nabla p = 0$), equation (3.3) becomes,

$$\nabla \times (\frac{\partial}{\partial t}(\boldsymbol{u} - \tilde{\boldsymbol{u}}) \cdot \boldsymbol{n}) = 0 \tag{3.5}$$

The time derivative can then be moved outside of the curl operation and equation (3.5) can be integrated with respect to time,

$$\nabla \times (\boldsymbol{u} - \tilde{\boldsymbol{u}}) \cdot \boldsymbol{n} = f(\boldsymbol{x}), \tag{3.6}$$

where $f(\boldsymbol{x})$ is some function of the space vector $\boldsymbol{x}$ and is effectively an integration constant. A value for $f(\boldsymbol{x})$ in equation (3.6) is found by applying the initial conditions to the problem. At $t = t_0$ it is known that $\boldsymbol{u} = \tilde{\boldsymbol{u}}$, therefore, at $t = t_0$ it must also be the case that $\nabla \times \boldsymbol{u} = \nabla \times \tilde{\boldsymbol{u}}$ and so $f(\boldsymbol{x}) = 0$ at t_0. Since $f(\boldsymbol{x})$ is a function of space only it is zero for all time t,

$$\nabla \times (\boldsymbol{u} - \tilde{\boldsymbol{u}}) \cdot \boldsymbol{n} = 0. \tag{3.7}$$

The next stage in obtaining the pressure correction equation uses the definition of a vector potential. This states that the curl of an irrotational vector can be defined as the grad of a scalar function, giving [36],

$$(\boldsymbol{u} - \tilde{\boldsymbol{u}}) \cdot \boldsymbol{n} = -\nabla\psi \cdot \boldsymbol{n} \tag{3.8}$$

The scalar value that has now been introduced, ψ, is defined as the pressure correction value. The penultimate stage in obtaining the pressure correction equation is to take the divergence of equation (3.8),

$$-\nabla^2\psi = \nabla \cdot \boldsymbol{u} - \nabla \cdot \tilde{\boldsymbol{u}}. \tag{3.9}$$

Finally the continuity equation can be applied to give the pressure correction equation,

$$\nabla^2\psi = \nabla . \tilde{\boldsymbol{u}} \tag{3.10}$$

It is through the pressure correction equation that continuity is enforced at each time step in the MAC algorithm.

After the pressure correction value ψ has been found by solving the above Poisson equation (equation (3.10)), the velocity and pressure must then be corrected. To correct the velocity and pressure, two further equations are required. Equation (3.8) can be rearranged to give the normal velocity correction equation,

$$u = \tilde{u} - \frac{\partial \psi}{\partial \boldsymbol{n}}. \tag{3.11}$$

Where u is the normal velocity component of $\boldsymbol{u}$. To obtain a similar equation for the pressure correction, equation (3.8) is substituted into equation (3.2),

$$p = \tilde{p} + \frac{\partial \psi}{\partial t}. \tag{3.12}$$

When the pressure and normal velocity have been corrected, the tangential velocity can be reconstructed from the normal velocity. The basic ideal behind reconstructing the tangential velocity is to assume that the Cartesian velocity vector is constant over a Delaunay element. The Cartesian velocity for the element is then defined as the average of the Cartesian velocities for each edge, these velocities at each edge are calculated from knowing the normal and tangential velocities. Since the tangential velocities are not known, a linear system can be constructed and inverted to find them. This brief overview will be expanded upon later, with the full formulation being given.

3.3 The Unstructured MAC Algorithm

The unstructured MAC algorithm decouples the calculation of velocity and pressure. As stated in the previous section the first step is to use a guess pressure to give an intermediate velocity. Then using the pressure correction equation as derived in the previous section a correction value for both the guess pressure and initial velocity can be found. To summarise the steps required for solution of the Navier-Stokes equations, the unstructured MAC algorithm as adapted from the cartesian mesh algorithm [13] is:

1. Let $\tilde{p}^m$ be an initial pressure, where m denotes the time step;

2. Calculate an intermediate normal velocity, $\tilde{u}^{m+1}$.

$$\frac{\partial \tilde{\boldsymbol{u}}^{m+1}}{\partial t} \cdot \boldsymbol{n} + \nabla(\boldsymbol{n}^m \cdot \boldsymbol{u}^m)\boldsymbol{u}^m = -\frac{\partial \tilde{p}^m}{\partial \boldsymbol{n}} - \frac{1}{Re}\frac{\partial}{\partial \tau}(\nabla \times \boldsymbol{u}^m)$$

the initial values for $\boldsymbol{u}^m$ are the initial velocity conditions for the problem being modelled.

3. Find the correction values by solving the Poisson problem,

$$\nabla^2 \psi^{m+1} = \nabla . \tilde{\boldsymbol{u}}^{m+1}$$

   subject to $\psi = 0$ on rigid boundaries or $\frac{\partial \psi}{\partial n} = 0$ on free boundaries.

4. Compute the corrected normal velocity,

$$u^{m+1} = \tilde{u}^{m+1} - \frac{\partial \psi^{m+1}}{\partial \boldsymbol{n}}.$$

5. Compute the corrected pressure,

$$p^{m+1} = \tilde{p}^{m+1} + \frac{\partial \psi}{\partial t}.$$

6. Calculate the tangential velocity.

7. Update the initial values. The guess pressure becomes the corrected pressure p^{m+1} and initial velocities $\boldsymbol{u}^m$ are set equal to $\boldsymbol{u}^{m+1}$

8. Iterate to the next time step $m + 1$ and repeat from step 2.

By following the steps outlined in the unstructured MAC algorithm a solution to the unsteady Navier-Stokes equations can be obtained by applying suitable discretisation to all the equations derived in the algorithm. The discretisation of each of the equations will now be described.

3.4 Momentum Equation Discretisation

To obtain a numerical solution for fluid flow problems the equations detailed in the unstructured MAC algorithm need to be approximated and evaluated at every discrete point, as given by a mesh of the domain. Therefore, a discrete form of each equation is required. The momentum equation shall be considered first, followed by the correction equations.

3.4.1 Explicit Discretisation

To describe the discretisation of the momentum equation, consider a general edge e, figure (3.3) details the variable layout around the edge e. The variables defined in

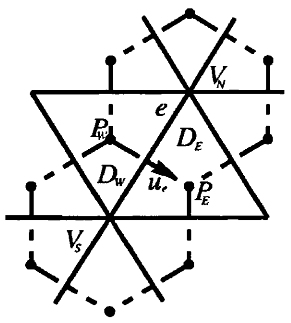


Figure 3.3: Staggered mesh variable layout for two Delaunay elements, D_E and D_W, which share a common edge e. The connected node which edge e links are the centres of the Voronoï cells V_N and V_S. The pressure variables P_E and P_W are located at the circumcentres of the Delaunay elements D_E and D_W respectively. The normal velocity variable u_e is located at the midpoint of edge e.

figure (3.3) are those required in forming a discretisation of the Navier-Stokes equations for the intermediate velocity, equation 3.1.

To begin with, the temporal term is discretised using an explicit forward difference scheme,

$$\frac{\partial \tilde{\boldsymbol{u}}}{\partial t} \cdot \boldsymbol{n}_e \approx \frac{u_e^{m+1} - u_e^m}{\Delta t}. \tag{3.13}$$

In equation (3.13), the velocity vector $\boldsymbol{u}$ is combined with the normal inner product to give $u_e = \boldsymbol{u} \cdot \boldsymbol{n}_e$, the normal velocity component to edge e. As before, the letter m is used to denote the time step and so $m + 1$ refers to the next time step, Δt is the time increment.

For the pressure term, a central difference about edge e is taken. If the pressure values at the Voronoï vertices either side of e are named P_E and P_W and the length of the Voronoï edge bisecting e is h_e^v, the discretisation is as follows,

$$\frac{\partial p}{\partial \boldsymbol{n}_e} \approx \frac{P_E^m - P_W^m}{h_e^v}. \tag{3.14}$$

Attention should be made to the direction of the derivative. In the formulation described, the derivative is always taken in the normal direction and so $\boldsymbol{n}_e$ will always point towards P_W for any edge.

The temporal and pressure terms are straight forward to discretise, they apply simply finite difference techniques. However, the viscous and convective terms, require more complex techniques. The discretisations for both these terms will require an integration over either a Voronoï cell or Delaunay element.

Formulating a discretisation for the viscous term can be split into two tasks; the approximation of the derivative of the curl of the velocity in the tangential direction of edge e; and the approximation of the curl of the velocity around the two Voronoï cells, connected by edge e. The transformation to the curl form as given in equation equation (2.6) has allowed this approach to be taken. The centre of the Voronoï cells, labelled V_N and V_S in figure (3.3), are the nodes linked by edge e. The distance between them is the length of edge e and is denoted by h^d_e, this is also the Delaunay cell edge length. The derivative of the curl in the tangential direction can be approximated using a central difference technique about the midpoint of edge e,

$$\frac{\partial}{\partial \boldsymbol{\tau}_e}(\nabla \times \tilde{\boldsymbol{u}}^m) \approx \frac{\nabla \times \tilde{\boldsymbol{u}}^m|_{V_N} - \nabla \times \tilde{\boldsymbol{u}}^m|_{V_S}}{h^d_e}. \tag{3.15}$$

To evaluate equation (3.15) fully, an approximation of the curl is required. The approximation can be obtained by an integration around the Voronoï cells. The transformation of the Viscous term to curl form is an essential part of the discretisation, without this a discretisation would not be so convenient to obtain on the given mesh framework. The curl term is considered as a volume integral, it can be converted back to the standard form by dividing by the volume of the cell. Stokes theorem can then be applied to convert the volume integral to a surface integral. First consider a general Voronoï cell V, Stokes theorem converts the volume integral over the Voronoï cell to the surface integral around the Voronoï cell,

$$\int_V \nabla \times \tilde{\boldsymbol{u}} \mathrm{d}\mathbf{x} = \int_{\partial V} \tilde{\boldsymbol{u}} \cdot \boldsymbol{\tau}_o \mathrm{ds}. \tag{3.16}$$

Stokes theorem states that the volume integral of the curl of a variable is equal to the surface integral of the dot product of the same variable with the anti-clockwise surface tangent.

Surface integrals are represented in discrete form as a sum of the variables stored at each edge of the cell. Each variable must be multiplied by its corresponding side length. This alone will not produce a correct solution as Stokes theorem assumes the tangent $\boldsymbol{\tau}_o$ is in the anti-clockwise direction. This is the equivalent to an outward normal from the intersecting Delaunay cell edge when a right-handed coordinate system is used. In the unstructured marker and cell staggered mesh layout, the predefined normals and tangents which the velocities lie in the direction of are not necessarily in the anti-clockwise direction. To compensate for this the velocity must be assigned a sign of one, if it is in the anti-clockwise direction or minus one otherwise. The integral is then discretised as,

$$\int_{\partial V} \tilde{\boldsymbol{u}} \cdot \boldsymbol{\tau}_o \mathrm{ds} \approx \sum_{i=1}^{s} u_i^m h_i^v (\boldsymbol{\tau}_o \cdot \boldsymbol{n}_i). \tag{3.17}$$

Note s is the number of sides of the Voronoï cell V and the h_i^v is the length of the ith Voronoï side. Voronoï edges lie perpendicular to Delaunay edges on an ideal mesh and so $\boldsymbol{\tau}_o$ is also the outward normal of the Delaunay edge. The product $\boldsymbol{\tau}_o \cdot \boldsymbol{n}_i$ is equal to 1 or -1, where $\boldsymbol{n}$ is the normal to the Delaunay edge, hence, defining a method of determining the direction of the velocity vector. The discretisation of the viscous term is not yet complete as the above sum needs to be divided by the area of the cell, A_V, giving the final discretisation of the curl as,

$$\nabla \times \tilde{\boldsymbol{u}} \approx \sum_{i=1}^{s} u_i^m h_i^v (\boldsymbol{\tau}_o \cdot \boldsymbol{n}_i)/A_V, \tag{3.18}$$

It is assumed that the value of the curl is constant over the Voronoï element. This assumption allows the value obtained in this discretisation to be assigned to the Voronoï cell centre. The techniques described for a general Voronoï cell V can be applied to the Voronoï cells V_N and V_S. The discretisation for the viscous term is then,

$$\frac{\partial}{\partial \boldsymbol{\tau}_e}(\nabla \times \tilde{\boldsymbol{u}}^m) \approx \frac{\sum_{i=1}^{sN} u_i^m h_i^v (\boldsymbol{\tau}_o \cdot \boldsymbol{n}_i)/A_{V_N} - \sum_{i=1}^{sS} u_i^m h_i^v (\boldsymbol{\tau}_o \cdot \boldsymbol{n}_i)/A_{V_S}}{h_e^d}, \tag{3.19}$$

where s_N is the number of sides around the Voronoï cell V_N and s_S is the number of sides around the Voronoï cell V_S, for a primary mesh formed of all trianglular elements these are both six.

The convective term follows a similar process but instead involves integration around two neighbouring Delaunay elements to approximate the divergence operator. An integral is taken around elements D_E and D_W shown in figure (3.3), since the value of the convective term is required at the midpoint of edge e, the weighted average of the two integrals is needed. To begin with, the average can be represented as,

$$\nabla \cdot (\boldsymbol{n}_e \cdot \tilde{\boldsymbol{u}})\tilde{\boldsymbol{u}}|_e \approx \frac{A_E}{A_E + A_W} \nabla \cdot (\boldsymbol{n}_e \cdot \tilde{\boldsymbol{u}})\tilde{\boldsymbol{u}}|_{D_E} + \frac{A_W}{A_E + A_W} \nabla \cdot (\boldsymbol{n}_e \cdot \tilde{\boldsymbol{u}})\tilde{\boldsymbol{u}}|_{D_W}, \tag{3.20}$$

where A_E and A_W are the areas for the Delaunay cells with centres D_E and D_W respectively. Again the value of a variable is constant over an element and so the integral result can be assumed to be located at the element centre. If a general Delaunay element D is considered, the divergence can be approximated by applying stokes theorem. For the divergence stokes theorem states that the volume integral of the divergence of a is equal to the surface integral of the dot product of a with the outward normal to the surface,

$$\int_{Vol} \nabla \cdot a \mathrm{d}\mathbf{x} = \int_{\delta Vol} a \cdot \boldsymbol{n}_o \mathrm{ds}. \qquad (3.21)$$

Substituting $(\boldsymbol{n}_e \cdot \boldsymbol{u})\boldsymbol{u}$ for a in equation 3.21 and dividing by the volume to convert back to a derivative the discretisation for the divergence of $(\boldsymbol{n}_e \cdot \boldsymbol{u})\boldsymbol{u}$ on the Delaunay cell D becomes,

$$\nabla \cdot (\boldsymbol{n}_e \cdot \boldsymbol{u})\boldsymbol{u}|_D \approx \frac{1}{Area}\left[\sum_{i=1}^{s} h_i^d(\boldsymbol{n}_o \cdot \boldsymbol{n}_i)u_i^m(\boldsymbol{n}_e \cdot \boldsymbol{u})_i\right]. \qquad (3.22)$$

Similarly to the viscous term the direction of the velocity needs to be determined. Stokes theorem for the convective term is applied to a div operator rather than a curl and so the outward normal $\boldsymbol{n}_o$ is assumed to be the direction of the velocity, rather than an anti-clockwise tangent. The velocity may not be in the outward normal direction and so a sign is determined using the product $(\boldsymbol{n}_o \cdot \boldsymbol{n}_i)$. Again, s represents the number of sides around the element in question, in this case, the element is a Delaunay triangle and so in the ideal case s is three. To complete the convective term, discretisation equation (3.22) can be applied to both D_E and D_W giving,

$$\begin{aligned}\nabla \cdot (\boldsymbol{n}_e \cdot \tilde{\boldsymbol{u}})\tilde{\boldsymbol{u}}|_P &\approx \frac{A_E}{A_E + A_W}\left[\sum_{i=1}^{s_E} h_i^d(\boldsymbol{n}_o \cdot \boldsymbol{n}_i)u_i^m(\boldsymbol{n}_e \cdot \boldsymbol{u})_i\right] \\ &+ \frac{A_W}{A_E + A_W}\left[\sum_{i=1}^{s_W} h_i^d(\boldsymbol{n}_o \cdot \boldsymbol{n}_i)u_i^m(\boldsymbol{n}_e \cdot \boldsymbol{u})_i\right],\end{aligned} \qquad (3.23)$$

where s_E is the number of sides around element D_E and s_W is the number of sides around D_W.

There is another challenge in the approximation of the convective term, the term $(\boldsymbol{n}_e \cdot \boldsymbol{u})_i$ its not straight forward to approximate. The velocity vector at an edge i is required in the same local coordinate system as the edge e. Therefore, the velocity and edge i must first be converted to a Cartesian coordinate system before conversion to the normal direction of edge e. Thus, the term can be expanded to $(\boldsymbol{n}_e \cdot \boldsymbol{u})_i = \boldsymbol{n}_e \cdot (u_i^m \boldsymbol{n}_i + v_i^m \boldsymbol{\tau}_i)$. Here, v_i^m is the tangential velocity for an edge i which is not calculated by the solution of the discrete momentum equation. This does not present a problem and the tangential velocity can successfully be reconstructed from the normal velocity, a technique that will be described later.

Full Explicit Momentum Equation Discretisation

For clarity, it is useful to view the momentum discretisation in its full form rather than term by term. Therefore the full discretisation for the momentum equation at the mid-point of an edge e is,

$$\begin{aligned} u_e^{m+1} &= u_e^m - \Delta t\Bigg(\frac{P_E - P_W}{h_e^v} + \frac{\nabla \times \boldsymbol{u}^m|_{V_N} - \nabla \times \boldsymbol{u}^m|_{V_S}}{h_e^d} \\ &+ \frac{A_E}{A_E + A_W}\nabla \cdot (\boldsymbol{n}_e \cdot \boldsymbol{u}^m)\boldsymbol{u}^m|_{D_E} \\ &+ \frac{A_W}{A_E + A_W}\nabla \cdot (\boldsymbol{n}_e \cdot \boldsymbol{u}^m)\boldsymbol{u}^m|_{D_W}\Bigg), \end{aligned} \tag{3.24}$$

where the curl and div operators are evaluated as given in the expressions (3.18,3.22). This discretisation is solved at every internal edge of the domain at every time step.

Boundary Conditions

The derived discretisations only apply to the interior points in the domain. To fully define the problem, the variables need to be defined on the boundaries of the domain. As previously discussed, the momentum equations are subject to the following boundary conditions: wall; moving wall; inflow and outflow. For a wall, moving wall or inflow condition values for the velocity can be specified. However, the velocity at an outflow boundary condition must be calculated.

The choice of boundary condition to apply will depend on the type of problem being modelled. Like the governing equations, the boundary conditions as discussed in section (2.3) will also require a suitable discrete representation. The numerical representation of the boundary conditions will now be discussed.

For a wall boundary, both velocity components are set to zero on any boundary edges that have this condition. This enforces the conditions that fluid cannot flow through or along the wall.

For a moving wall, the velocity components are specified to reflect the required velocity of the wall. Pressure is unknown on wall boundaries but since pressure is not stored at the boundaries, it does not need to be found or specified.

An inflow boundary condition is straight forward to implement, like the moving wall both velocity components are set. For an inflow both velocity components are set depending on the required direction of the flow. Pressure is not known at this boundary, although it is often necessary to specify a reference pressure for one boundary element on. When a reference pressure is specified instabilities in the unstructured MAC method are avoided.

The inflow partner boundary condition, the outflow, is more difficult to implement as it requires the velocity to be calculated on the boundary. The chosen method for the outflow is to calculate the normal velocity on the boundary using the momentum equation discretisations. Viscous effects are neglected, however, special consideration needs to be taken for the pressure and convective terms, as they require knowledge of values that lie outside the domain, as illustrated in figure (3.4). For the pressure term,

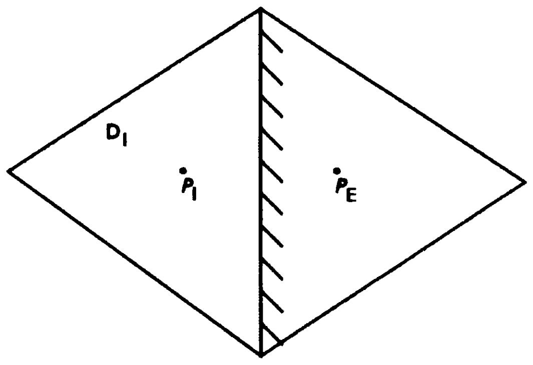


Figure 3.4: An element that lies on a boundary with fictitious boundary element outside the domain

the momentum equation requires knowledge of the pressure just outside the domain and so a Neumann condition,

$$\frac{\partial p}{\partial \boldsymbol{n}} = 0$$

is assumed. The Neumann condition allows the pressure exterior of the domain to be assigned the value of the interior pressure, i.e. $P_I = P_E$. A similar problem arises with the convective term. The discretisations require the divergence around an element that does not exist i.e. the divergence is required around the element that has P_E and its centre, in figure (3.4). In this case the convective term is not calculated as an average of the divergence over two elements. Instead it is only calculated over the element that exists, the element labelled D_I in figure (3.4). The convective term for an outflow boundary edge then becomes,

$$\nabla \cdot (\boldsymbol{n}_e \cdot \boldsymbol{u})\boldsymbol{u}|_e \approx \nabla \cdot (\boldsymbol{n}_e \cdot \boldsymbol{u})\boldsymbol{u}|_{D_I}$$

The boundary conditions for the velocity need to be applied during the calculation of the intermediate velocity.

A consequence of the marker and cell pressure correction procedure is that the pressures calculated at a time step, are pressure variations from a set reference pressure. Therefore it is necessary to set a value for the reference pressure at one of the boundaries. An outflow is a good boundary to specify the reference pressure, in doing so it was observed that convergence of the routine was often improved. An alternative is to specify a reference pressure at one of the boundary elements.

Voronoï Cells Adjacent to a Boundary

To fully complete the description of the discrete problem for the unstructured MAC method, details on how to construct Voronoï cells at the domain boundaries need to be given. Delaunay elements are fully aligned with the boundary because the Voronoï cells are constructed by joining Delaunay circumcentres, the definition of Voronoï cells that involve a boundary Delaunay element is ambiguous. Therefore, a method of defining a Voronoï cell on the boundary is required. There are potentially two options for defining these elements. The first of which, would be to try and construct a full Voronoï cell. This is not a desirable approach as it involves edges that do not exist in the mesh, therefore, requiring some method of defining variables at these locations, see figure (3.5). The alternative approach is to define the boundary Voronoï cells so that they follow the boundary edge, see figure (3.6). It is this approach that is applied in the unstructured MAC algorithm. The anticlockwise tangents around the boundary Voronoï cells, that

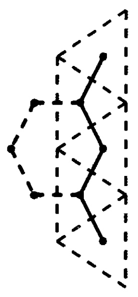


Figure 3.5: A full Vornoï cell does not exist on the boundary

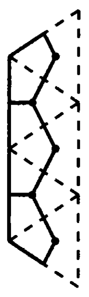


Figure 3.6: Construction of Voronoï cells on the boundary of the domain

occur in the viscous term discretisation, now include tangents along Delaunay edges not just Voronoï edges. A consequence of this is that the curl discretisation now includes a tangential velocity as well as the normal. In equation (3.17) it is assumed that the tangent around the Voronoï cell $\boldsymbol{\tau}_o$ is the normal to a Delaunay element. However, for Voronoï cells that lie on the boundary, the tangent $\boldsymbol{\tau}_o$ is now the tangent to a Delaunay element, meaning that $\boldsymbol{u} \cdot \boldsymbol{\tau}_o = v$ instead of u. For these boundary edges there is still a contribution form the normal velocity as the short edge that connects the circumcentre to the domain boundary lies in the normal direction.

Time Step Constraints

The time stepping scheme chosen for the algorithm is an explicit Euler method. This method is only stable under certain time step size conditions. If a time step Δt is too large then the unstructured MAC scheme will not produce a stable result.

The time step constraint for the unstructured MAC algorithm was originally adapted from the time step of the cartesian mesh version of the algorithm, [13]

$$\Delta t < \min\left(\frac{Re}{2}\frac{(h^d_{min})^2(h^v_{min})^2}{(h^d_{min})^2+(h^v_{min})^2}, \frac{h^d_{min}}{u_{min}}, \frac{h^v_{min}}{v_{min}}\right), \tag{3.25}$$

where h^d_{min} is the minimum Delaunay edge length for the domain, h^v_{min} the minimum Voronoï edge length, u_{min} the minimum normal velocity and v_{min} the minimum tangential velocity. The first condition is the restriction due to the Viscous term. The remaining two conditions do not allow fluid to flow through more than one element each time step, i.e. they are the convective time step constraints. The time step here will be defined as the minimum values time step.

The condition in equation (3.25) was later found to produce much smaller time steps than necessary, especially when using more irregular meshes. This led to a new selection method being devised. The new method requires a time step to be calculated for each edge. The lengths and velocities for that edge are used to calculated the time step rather than using global minimum lengths and velocities, as in equation (3.25). The conditions themselves are the same and are reformed as,

$$\Delta t_e < \min\left(\frac{Re}{2}\frac{(h^d_e)^2(h^v_e)^2}{(h^d_e)^2+(h^v_e)^2}, \frac{h^d_e}{u_e}, \frac{h^v_e}{v_e}\right). \tag{3.26}$$

The minimum of the three conditions is found to give the time step Δt_e, which refers to the time step size for the particular edge e, h^d_e, h^v_e, u_e and v_e are the respective Delaunay and Voronoï edge lengths and normal and velocity components at edge e. The minimum of all the Δt_e is then the single time step size for the domain,

$$\Delta t \leq \min\left(\Delta t_e\right).$$

The time step in this case is named the minimum edge time step.

Using either the minimum values time step or the minimum edge time step will produce a valid time step size. The minimum edge time step has the potential to use a larger time step. The use of a larger time step is desirable as an efficient algorithm is being derived. Less time steps will be required to reach a specific time if larger time steps are used.

Both methods are adaptive and can be recalculated each iteration to allow the largest possible Δt to be used. The minimum values of the velocity component may change each iteration. By allowing the time step calculation to be adaptive, means the largest time step possible for each iteration will be used. If adaptive time stepping is not applied, a smaller time step than necessary may be used for some iterations.

The time step size calculated by either method should be multiplied by a safety factor, the Courant-Friedrichs-Lewy (CFL) number. The CFL condition is a restraint on the time step size when using explicit time stepping schemes. For an incompressible flow,

$$\frac{u\Delta t}{\Delta x} < C$$

is usually applicable [10], where C is the CFL number and Δx is the change in the x position. Rearranging gives,

$$\Delta t < C\frac{\Delta x}{u}$$

which is equivalent to the convective time step conditions. It is therefore viable to multiply the obtained Δt by C. Formally for hyperbolic partial differential equations (PDEs) $0 < C \leq 1$ [77]. However, the incompressible Navier-Stokes equations are not fully hyperbolic and using the time step conditions applied here there is the potential for $C > 1$. Since this is the case, C can be viewed as a scaling factor for the time step, its value will depend on the problem being modelled.

Equation (3.26) gives the means to determine a time step if a local time stepping procedure were used. Local time stepping can give faster solutions for steady state flows. The momentum equations are advanced to the next iteration using the time step size for each individual edge. In a steady state solution, time accuracy is not required and so each edge can be advanced in time independently until convergence has been reached.

3.4.2 Implicit Discretisation

The alternative to using an explicit discretisation is to use an implicit one. An Implicit discretisation of the momentum equations differs from the explicit discretisations in that all terms involving the dependent variable are calculated at the same time level. This means that equation (3.1) becomes,

$$\frac{\partial \tilde{\boldsymbol{u}}}{\partial t} \cdot \boldsymbol{n} + \nabla(\boldsymbol{n} \cdot \tilde{\boldsymbol{u}})\tilde{\boldsymbol{u}} = -\frac{\partial \tilde{p}}{\partial \boldsymbol{n}} - \frac{1}{\mathrm{Re}}\frac{\partial}{\partial \tau}(\nabla \times \tilde{\boldsymbol{u}}). \tag{3.27}$$

All terms except the pressure term involve the unknown intermediate velocity. Over the domain this produces a set of simultaneous equations which can be solved by a matrix inversion.

An implicit method is desirable since it allows a larger value for the time step. A larger time step is advantageous, provided the time step is large enough that any extra computation cost in performing the matrix inversion is negated by fewer time steps being required. Using an implicit method becomes a necessity when using stretched boundary layer meshes. In the case of meshes with stretched boundary layers the explicit time step size is very small and so an unsteady solution is not obtainable in a suitable amount of time.

In an implicit discretisation, the temporal term is discretised in the same manner as the explicit time stepping scheme, the difference is that, in an implicit time stepping scheme the remaining terms are evaluated at the new time step $m+1$ rather than m. The discretisations for the other terms are formed in the same way, however, they would now involve the intermediate velocity, the unknown value.

There are many types of implicit time stepping schemes, two of which will be considered here, as described in [78]. The first to consider is the forward Euler method. As the governing equations of this work are the incompressible Navier-Stokes equations, let u represent the unknown velocity value, $\mathrm{vis}(t, u)$ represents the viscous term that depends on time and u, $\mathrm{con}(t, u)$ represents the convective term which depends on time and u. $\mathrm{pre}(t)$ represents the pressure which only depends on time. An implicit Euler method for the Navier-Stokes equations can then be written as,

$$u^{m+1} = u^m + \Delta t \left[\mathrm{vis}(t, u^{m+1}) + \mathrm{con}(t, u^{m+1}) + \mathrm{pre}(t)\right]. \tag{3.28}$$

An alternative scheme that could be implemented is the Crank-Nicolson method. The Crank-Nicolson method is unconditionally stable for linear systems and so for non-linear systems it should still allow the use of a larger time step than the forward Euler method [47]. In the Crank-Nicolson method, u, is required at both the new time step and the previous time step. The Crank-Nicolson method is given as,

$$\begin{aligned} u^{m+1} = u^m + \Delta t \big[& 0.5\left(\text{vis}(t, u^{m+1}) + \text{con}(t, u^{m+1})\right) + \\ & 0.5\left(\text{vis}(t, u^m) + \text{con}(t, u^m)\right) + \text{pre}(t)\big] \end{aligned} \tag{3.29}$$

The Crank Nicolson method is a generalisation, the general case allows a weight of θ for the terms evaluated at time step $m+1$ and a weight of $(1-\theta)$ for the terms evaluated at the time step m. The case where $\theta = 0.5$ is named the Crank-Nicolson method and is the only choice that is unconditionally stable for linear problems [78].

A Semi-Implicit Discretisation

In the unstructured MAC framework, implementation of the implicit time stepping routine as described in the previous section, is considered by the author of this thesis to be a computationally inefficient procedure. Prior identification of all edges involved in every discrete calculation of the viscous and convective terms is required for every edge. To elaborate on this, an implicit viscous term formulation would require every edge that forms each Voronoï cell to be stored, coupled with the signs that determine the correct direction of the edge i.e. whether a Voronoï edge has an anti-clockwise or clockwise tangent. Similarly, implicit formulation of the convective term would require knowledge of all sides that form each Delaunay cell and the corresponding signs. A further challenge in using a full implicit scheme is in determining which are the common edges required in both convective and viscous term calculations. It is feasible that efficient techniques can be devised to construct a fully implicit time stepping scheme within the unstructured MAC framework. However, due to time constraints this author has chosen to investigate another avenue.

An alternative to the fully implicit time stepping routine is to use a semi-implicit time stepping routine. In a semi-implicit time stepping routine, either the viscous or convective term is treated implicitly, the other is treated explicitly. The time step restrictions given in equations (3.25) and (3.26), show that it is the viscous time step that has the potential to be most restrictive. This restriction is especially prominent when small Voronoï edges due to the use of stretched boundary elements occur in the mesh.

The use of stretched meshes is of interest as a high resolution mesh is required around an immersed body to fully capture viscous effects in flows with large Reynolds numbers. This high resolution is only required in the normal direction out from the immersed body and not in the tangential direction. Stretched meshes allow a computational saving as the resolution is only higher in the normal direction.

The viscous term is most restrictive on the time step so a semi-implicit formulation introduced in this work will treat the viscous term implicitly and the convective term explicitly. Many successful attempts at using a semi-implicit discretisation have been reported in the literature [47, 46, 45, 41]. All these works implement an implicit treatment of the viscous term, therefore, it is adopted as a suitable alternative to the fully implicit time stepping scheme in the unstructured MAC solver.

The formulation of the discrete Navier-Stokes equations for the intermediate velocity can be modified for the semi-implicit case too,

$$\frac{\partial \tilde{u}}{\partial t} \cdot \boldsymbol{n} + \nabla(\boldsymbol{n} \cdot \boldsymbol{u})\boldsymbol{u} = -\frac{\partial \tilde{p}}{\partial \boldsymbol{n}} - \frac{1}{\mathrm{Re}}\frac{\partial}{\partial \tau}(\nabla \times \tilde{\boldsymbol{u}}). \tag{3.30}$$

The velocity in the convective term is the initial velocity and no longer the intermediate velocity. The same discretisation as for the explicit case are still applied. The viscous term now includes velocities that are now the unknown intermediate velocities. The full discrete momentum equation for an edge e, using a semi-implicit time discretisation is,

$$\begin{aligned} u_e^{m+1} &- \Delta t\left(\frac{\nabla \times \boldsymbol{u}^{m+1}|_{V_N} - \nabla \times \boldsymbol{u}^{m+1}|_{V_S}}{h_e^d}\right) = u_e - \Delta t\left(\frac{P_E - P_W}{h_e^v}\right. \\ &+ \frac{A_E}{A_E + A_W}\nabla \cdot (\boldsymbol{n}_e \cdot \boldsymbol{u}^m)\boldsymbol{u}^m|_{D_E} \\ &+ \left.\frac{A_W}{A_E + A_W}\nabla \cdot (\boldsymbol{n}_e \cdot \boldsymbol{u}^m)\boldsymbol{u}^m|_{D_W}\right). \end{aligned} \tag{3.31}$$

All unknowns are located on the left hand side of equation (3.32) and all the known values on the right. The assembled set of discrete equations form a matrix system. This matrix system has a symmetric structure but the values are not symmetric, thus the matrix is unsymmetric.

Symmetry can be retained in the values by using a uniform triangular mesh, a problem that is easier to solve as techniques for solving systems with symmetric matrices are more widely available. However, limiting to meshes that produce this symmetric matrix would severely limit the capabilities of the implicit solver and so an efficient non-symmetric system solver is required.

The forward Euler method, equation (3.28) and Crank-Nicolson method, equation (3.29), can be modified, to apply in the semi-implicit framework. The semi-implicit forward Euler method is,

$$u^{m+1} = u^m + \Delta t \left[\text{vis}(t, u^{m+1}) + \text{con}(t, u^m) + \text{pre}(t)^m \right], \tag{3.32}$$

and the semi-implicit scheme using the Crank-Nicolson method becomes,

$$u^{m+1} = u^m + \Delta t \left[0.5\text{vis}(t, u^{m+1}) + 0.5\text{vis}(t, u^m) + \text{con}(t, u^m) + \text{pre}^m \right]. \tag{3.33}$$

The implementation of the boundary conditions remains the same as for the explicit case. This is due to viscous effects being neglected at outflow boundaries and so viscous term calculation is not needed at any of the boundaries. Therefore, no implicit boundary implementation is necessary. The boundaries do have some effect on the viscous term calculation, when a Voronoï cell lies adjacent to a boundary as in figure (3.6). These boundary Voronoï cells introduce an explicit and implicit part to the viscous term, the explicit part coming from the known tangential velocity.

The semi-implicit discretisation is easily incorporated into the unstructured marker and cell algorithm. The only change is in the calculation of the intermediate velocity, instead of using the explicit scheme, the intermediate velocity is found using the implicit scheme.

3.5 Pressure Correction Equation

3.5.1 Discretisation

There is no equation for pressure in an incompressible flow and so the pressure correction equation (3.10) must be solved. Therefore, a suitable discretisation is required for the Poisson equation

$$\nabla^2 \psi = \nabla \cdot \tilde{\boldsymbol{u}}$$

on unstructured meshes. To obtain the discretisation, a similar process to that of the momentum equation discretisations is adopted.

First consider the left hand side of equation (3.10). The term

$$\nabla^2 \psi \tag{3.34}$$

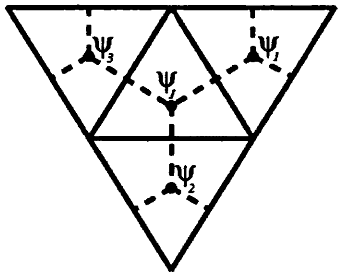


Figure 3.7: Variable configuration for pressure correction values

is integrated over the Delaunay cell D with area A_D. ∇^2 is the divergence of the gradient and can be split so that

$$\nabla^2\psi = \nabla \cdot \nabla\psi.$$

Stokes theorem is then applied to convert the volume integral to a surface integral,

$$\int_D \nabla \cdot \nabla\psi \mathrm{dx} = \int_{\partial D} \nabla\psi \cdot \boldsymbol{n}\mathrm{ds} \tag{3.35}$$

where $\boldsymbol{n}$ is the outward normal of the Delaunay cell D. A simplification can then be made so that,

$$\int_{\partial D} \nabla\psi \cdot \boldsymbol{n}\mathrm{ds} = \int_{\partial D} \frac{\partial\psi}{\partial\boldsymbol{n}}\mathrm{ds} \tag{3.36}$$

Equation (3.36) indicates that to form a discrete approximation of the term (3.34) the gradient of ψ in the outward normal direction for all edge of the Delaunay element is required. The derivative is approximated at a Delaunay edge using a central difference approximation,

$$\frac{\partial\psi}{\partial\boldsymbol{n}}|_i = \frac{\psi_i - \psi_J}{h_i^v} \tag{3.37}$$

where i identifies the neighbouring ψ value in the normal direction out of the element. h_i^v is the length of the Voronoï edge which joins the two ψ values. Equation 3.37 can be coupled with equation 3.36 to give,

$$\int_{\partial D} \frac{\partial\psi}{\partial\boldsymbol{n}}\mathrm{ds} \approx \sum_{i=1}^{3} \frac{(\psi_i - \psi_J)h_i^d}{h_i^v}/A_D. \tag{3.38}$$

Using the variable layout in figure (3.7) ψ_J refers to the current element and ψ_i where $i = 1, 2, 3$, for an element with three edges, refers to the surrounding elements. The same technique of applying Stokes theorem to the divergence operator in the convective term discretisation is applied to the right hand side,

$$\int_D \nabla \cdot \boldsymbol{u} \mathrm{d}\mathrm{x} = \int_{\partial D} \boldsymbol{u} \cdot \boldsymbol{n} \mathrm{d}\mathrm{s} \approx \sum_{i=1}^{3} u_i^m h_i^d (\boldsymbol{n}_i \cdot \boldsymbol{n}_o)/A_D. \tag{3.39}$$

When equation (3.38) and equation (3.39) are combined the area cancels out giving the discrete equations as,

$$\sum_{i=1}^{3} \frac{(\psi_i - \psi_J)h_i^d}{h_i^v} = \sum_{i=1}^{3} u_i^m h_i^d (\boldsymbol{n}_i \cdot \boldsymbol{n}_o), \tag{3.40}$$

Terms on the right hand side of equation (3.40) are known and so a symmetric implicit system with a vector of unknown values containing all ψ_i is produced. An implicit system has various solution techniques some of which will be discussed later.

Again the number of edges that constructs and element can be increased and the discretisations will still apply. The sums in equation (3.40) will be up to the number of edges an element has rather than to three.

3.5.2 Boundary conditions

Boundary conditions for the velocity and pressure are not the only ones that need to be considered. There is also the need for boundary conditions for ψ, the pressure correction value. Not only does this allow the solution of the Poisson equation to be possible but it also applies boundary effects to the pressure through the correction stage. There are two types of boundary condition for ψ depending whether the pressure is known or unknown. If the pressure is known, then there is no need to correct it, therefore, the correction value is zero. So for an outflow boundary, as pressure is known to be equal to the exterior value, $\psi = 0$. If the pressure is unknown on the boundary then ψ is also unknown on a boundary and can be found by solving the Poisson equation discretisation for the boundary edge. A value for ψ is required outside the domain and so a Neumann boundary condition, $\frac{\partial \psi}{\partial n} = 0$ is applied. This condition is applicable to any wall condition and an inflow condition. The boundary conditions relating to ψ are applied during the implicit solution process of the pressure correction equation.

Coupling the required boundary conditions for a particular problem with the discretisations for the internal domain ensures a well defined problem from which accurate numerical results can be obtained.

3.6 Correcting the Pressure and Velocity

After the correction value ψ has been calculated, the intermediate velocity and the initial pressure can be corrected. Correcting the pressure and velocity is achieved by discretising and solving equations (3.12) and (3.11) for every element or edge respectively. For the velocity, the discretisation for the edge e as shown in figure (3.3) is,

$$u_e^{m+1} = \tilde{u}_e^{m+1} - \frac{\psi_W - \psi_E}{h_e^v},$$

where ψ_W is at the same location as P_W and ψ_E as P_E.

The pressure correction is corrected by dividing ψ in equation (3.12) by the time step Δt to give p^{n+1} for an element J as,

$$p_J^m = \tilde{p}_J^m + \frac{\psi}{\Delta t}.$$

It is also important to ensure that the boundary conditions are satisfied correctly after the correction stage. If the velocity is known at the boundary then there is no need to correct its value, however, in certain outflow cases, usually when the viscous term is neglected or when the boundary is not located far enough away from fluid effects, the velocity at the outflow may also need correcting. The velocity is corrected in the same manner as for the interior edges, by applying equation (3.11). The gradient requires a value of ψ outside the computational domain, like the pressure gradient a Neumann boundary condition is applied so that $\psi_I = \psi_E$, where I denotes the variable on the interior of the domain and E on the exterior.

3.7 Tangential Velocity

The discretisations given above only solve for the normal velocity component to an edge. However, the tangential velocity component is not only required to fully approximate the normal velocity component in the convective term but also whenever the full velocity solution is required e.g. for solution output. From equation (2.8) it can be seen that given the co-volume mesh layout previously described, that a discretisation for the tangential component would be difficult. To overcome this difficulty the tangential velocity can be reconstructed from the normal velocity.

Two methods are given in [64]. The chosen method detailed here is considered the most accurate. The method first assumes that the Cartesian velocity vector $\boldsymbol{w}$ =

$(w_1, w_2)^T$, is constant over a cell. Recalling that u_i and v_i are the normal and tangential velocity components for a particular edge i. The value of $\boldsymbol{w}$ for a Delaunay element with three sides can be calculated as,

$$\boldsymbol{w} = \frac{1}{3}\sum_{i=1}^{3}(u_i\boldsymbol{n}_i + v_i\boldsymbol{\tau}_i).$$

This is calculating the cell Cartesian velocity as being an average of all Cartesian velocities on its surrounding sides. In the above sum, it is assumed all cell sides are of equal length. However, it is possible to change this sum to a weighted average, perhaps producing more accurate results on much more irregular meshes particular for cases where stretched elements are used.

To obtain the tangential velocity component v_i for a particular edge, the dot product of the cell Cartesian velocity must be taken with the edge tangent. The calculation of the tangential velocity must be treated on an element by element basis. This is due to the Cartesian velocity requiring the unknown tangential velocities for every edge that constructs each element. By taking the dot product of $\boldsymbol{w}$ with tangent $\boldsymbol{\tau}$ for each edge that constructs the same element, a system of three equations is produced so that,

$$v_i = \boldsymbol{w}\cdot\boldsymbol{\tau}_i, \qquad i = 1, 2, 3. \tag{3.41}$$

These are three simultaneous equations which can be solved to find each v_i. In the unstructured MAC algorithm these equations are solved by rearranging the above equations to form the following linear system,

$$\begin{pmatrix} \frac{2}{3} & -\frac{\boldsymbol{\tau}_2\cdot\boldsymbol{\tau}_1}{3} & -\frac{\boldsymbol{\tau}_3\cdot\boldsymbol{\tau}_1}{3} \\ -\frac{\boldsymbol{\tau}_1\cdot\boldsymbol{\tau}_2}{3} & \frac{2}{3} & -\frac{\boldsymbol{\tau}_3\cdot\boldsymbol{\tau}_2}{3} \\ -\frac{\boldsymbol{\tau}_1\cdot\boldsymbol{\tau}_3}{3} & -\frac{\boldsymbol{\tau}_2\cdot\boldsymbol{\tau}_3}{3} & \frac{2}{3} \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} \frac{1}{3}\left[u_1\boldsymbol{n}_1\cdot\boldsymbol{\tau}_1 + u_2\boldsymbol{n}_2\cdot\boldsymbol{\tau}_1 + u_2\boldsymbol{n}_3\cdot\boldsymbol{\tau}_1\right] \\ \frac{1}{3}\left[u_1\boldsymbol{n}_1\cdot\boldsymbol{\tau}_2 + u_2\boldsymbol{n}_2\cdot\boldsymbol{\tau}_2 + u_3\boldsymbol{n}_3\cdot\boldsymbol{\tau}_2\right] \\ \frac{1}{3}\left[u_1\boldsymbol{n}_1\cdot\boldsymbol{\tau}_3 + u_2\boldsymbol{n}_2\cdot\boldsymbol{\tau}_3 + u_3\boldsymbol{n}_3\cdot\boldsymbol{\tau}_3\right] \end{pmatrix}$$

The system is solved for every element and so two values for the tangential velocity at every non boundary edge will be calculated. Therefore, the tangential velocity at each edge is the average of the two values.

The process has been described for elements constructed of three edges. The method is applicable to any element with any number of edges, s. The system to be solved to find the tangential velocities will then be an $s \times s$ system.

3.8 Alternative convective term

In the literature, there is another possible option for the discretisation of the connective term [66]. The scheme proposed in the previous sections is a central differencing scheme. The alternative discretisation can either be a central differencing or an upwind differencing scheme.

Using the diagram in figure (3.8) the variable layout is described, P is the midpoint of the current edge, Q' and R' are the points of intersection of the line in the direction of $\boldsymbol{u}$ with the element edges. R' lies on the element edge that is in the normal direction, whereas, Q' lies on the element edge in the opposite direction. The velocity vectors

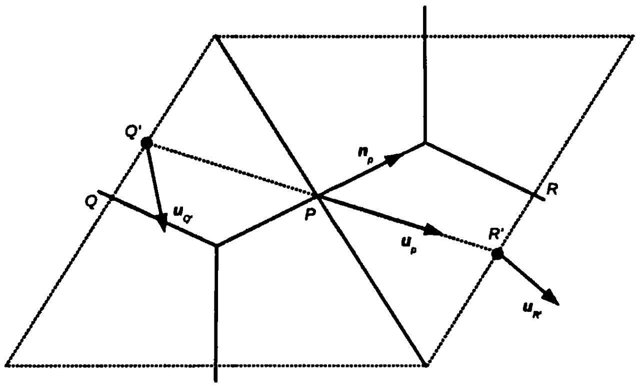


Figure 3.8: Variable layout for the alternative convective term

$\boldsymbol{u}_{Q'}$ and $\boldsymbol{u}_{R'}$ are located at these points respectively. In this case, $\boldsymbol{u}$ is referring to the Cartesian velocity vector and so the necessary transformation from the local normal and tangential components are required.

One challenge with this method, is that velocity values are not known at Q' and R' but are known at the intersection point with the Voronoï edges, Q and R. It is therefore necessary to make the assumption that velocities along an edge are constant along that edge. This allows the velocities at the midpoints to be used in place of $\boldsymbol{u}_{Q'}$ and $\boldsymbol{u}_{R'}$, defining these as $\boldsymbol{u}_Q$ and $\boldsymbol{u}_R$.

The first step towards formulating the alternative convective term discretisation is to apply the continuity equation to make the transformation,

$$[\nabla \cdot (\boldsymbol{u}\boldsymbol{u})] \cdot \mathbf{n} = [\nabla(\boldsymbol{u} \cdot \mathbf{n})] \cdot \boldsymbol{u}.$$

If $\boldsymbol{u} \neq 0$ then the above expression can be transformed to,

$$[\nabla \cdot (\boldsymbol{u}\boldsymbol{u})] \cdot \boldsymbol{n} = \left[\frac{\partial}{\partial u}(\boldsymbol{u} \cdot \boldsymbol{n})\right] |\boldsymbol{u}|, \tag{3.42}$$

by making use of the directional derivative, where $|\boldsymbol{u}|$ is the Euclidean length of the vector. The expression (3.42) can be discretised using either a central or upwind difference scheme. The central difference discretisation is given as,

$$\left\{\frac{\partial}{\partial u}[\boldsymbol{u}(P) \cdot \boldsymbol{n}_P]\right\} |\boldsymbol{u}(P)| \approx \pm \frac{\boldsymbol{u}_{R'} \cdot \boldsymbol{n}_P - \boldsymbol{u}_{Q'} \cdot \boldsymbol{n}_P}{|R' - Q'|} |\boldsymbol{u}_P|. \tag{3.43}$$

The upwind scheme can be given as,

$$\left\{\frac{\partial}{\partial u}[\boldsymbol{u}(P) \cdot \boldsymbol{n}_P]\right\} |\boldsymbol{u}(P)| \approx \begin{cases} \frac{\boldsymbol{u}_P \cdot \boldsymbol{n}_P - \boldsymbol{u}_{Q'} \cdot \boldsymbol{n}_P}{|P - Q'|} |\boldsymbol{u}_P| & \text{if } \boldsymbol{u}_p \cdot \boldsymbol{n}_p \geq 0, \\ \frac{\boldsymbol{u}_P \cdot \boldsymbol{n}_P - \boldsymbol{u}_{R'} \cdot \boldsymbol{n}_P}{|P - R'|} |\boldsymbol{u}_P| & \text{if } \boldsymbol{u}_p \cdot \boldsymbol{n}_p < 0. \end{cases} \tag{3.44}$$

All other terms in the Navier-Stokes equation are discretised in the manner previously detailed. This alternative convective term discretisation is stated as only first order accurate [66] and so it is still preferable to use the original discretisation.

3.9 Solvers for Implicit Systems

The use of an implicit time stepping routine means that a suitable solver for the matrix system is required. An implicit system solver is also required in the explicit case for the solution of the pressure correction equation (3.12). The discretisations for both cases produce a set of simultaneous equations which can be arranged into a matrix system, **A**, to be solved for all unknowns, **x**,

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{3.45}$$

where **A** is the $n \times n$ matrix,

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix}$$

The unknown values construct the vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ and the known values construct the vector $\mathbf{b} = (b_1, b_2, \ldots, b_n)^T$. Equation (3.45) can then be solved by inverting matrix $\mathbf{A}$ and multiplying by the inverted matrix so that,

$$\mathbf{A}^{-1}\mathbf{b} = \mathbf{x} \tag{3.46}$$

The standard technique to solve equation (3.45) is Gaussian elimination. However, the techniques applied in CFD result in **A** being a very large matrix. Gaussian elimination is an adequate technique for small matrices but when applied to large matrices it is very inefficient. Therefore, alternative faster solution methods for the system are required.

There are various methods available to solve an implicit system. There are two distinctive classes of solution methods, direct and iterative, both of which have advantages and disadvantages. It is important that the chosen method be an efficient option and so several are now described.

3.9.1 Direct Methods

Direct methods invert the matrix **A** directly and so the solution they produce to the system in equation (3.45) is exact, a distinct advantage of using a direct method. A disadvantage of using such methods is that they tend to be computationally slow (Gaussian elimination being a direct method). However, faster direct methods such as the frontal method [79] and multi frontal method [80] have been developed and allow the fast solution of the systems. Many of the Gaussian elimination processes apply to other direct solution methods and so the concepts it applies will now be detailed.

The general summation for each new entry to **A** and **b** in Gaussian elimination can be given as,

$$a_{i,j} = 0 \qquad i > k, j \leq k \tag{3.47}$$

$$a_{i,j} = a_{i,j} - (a_{i,k}/a_{k,k})a_{k,j}, \qquad i, j > k \tag{3.48}$$

$$b_i = -(a_{i,k}/a_{k,k})b_k, \qquad i > k \tag{3.49}$$

The process is carried until the nth row forms the

$$a_{n,n}x_n = b_n. \tag{3.50}$$

The remaining x_n can then be found by back substitution. Although Gaussian elimination is a well known technique, it has been introduced here as the faster direct methods require knowledge of the equations that describe the method.

Further to Gaussian elimination, the next direct method to be introduced is LU decomposition. LU decomposition is still a computationally slow technique but for small matrix inversion it is a viable method to implement. Again, LU decomposition has been deemed an unsuitable option for the inversion of large scale systems such as those that form the pressure correction equation. However, some of its principles are useful in other techniques and so the method is now described. The principle of LU decomposition are that its that matrices **L** and **U** are constructed such that,

$$\mathbf{LU} = \mathbf{A}. \tag{3.51}$$

L is a lower triangular matrix and **U** an upper triangular matrix. The decomposition of **A** into the lower and upper triangular matrices, equation (3.51), can be substituted into the matrix system equation (3.45),

$$(\mathbf{LU})\mathbf{x} = \mathbf{L}(\mathbf{Ux}) = \mathbf{b}$$

A solution is obtained by first solving for a vector **y**, where $\mathbf{Ux} = \mathbf{y}$, so that,

$$\mathbf{Ly} = \mathbf{b}$$

Once **y** has been found then **x** can be found,

$$\mathbf{Ux} = \mathbf{y}$$

Since **L** and **U** are triangular matrices, the systems above can be solved by forward and back substitution. Finding the matrices **L** and **U**, which can be accomplished via Crouts algorithm, it is the most costly part of LU decomposition. A desirable consequence of LU decomposition is that the vector **b** can change and as long as **A** remains the same, **L** and **U** do not need to be found again. To then solve for a new right hand side, only the forward and back substitution stages need to be implemented. For a detailed description of LU decomposition, see [77] and for a Fortran implementation, see [81].

The matrices involved with the implementation of the marker and cell method are sparse matrices and for efficiency, a direct solver that takes advantage of this property is desirable. Options for solving sparse matrix systems include the frontal and multi-frontal methods. The frontal method was originally devised for the solution of finite element problems by Irons as mentioned in [80]. The basic idea of which, is to split the matrix into a sum of smaller sub matrices, in the finite element case there exists a sub-matrix for each element. The technique can be applied to a general case with a subdivision of the matrix being defined, using each row as a subdivision. To explain the frontal method **A** can be defined as a sum of sub matrices [80],

$$\mathbf{A} = \sum_{l} \mathbf{A}^{[l]}$$

The sum can be assembled for each element as,

$$a_{i,j} := a_{i,j} + a_{i,j}^{[l]}$$

and is fully summed when all elements that need to have contributed to the sum. The method then applies the basic operation of Gaussian elimination, equation (3.48), before all the assemblies are complete. The eliminations can then be applied to the sub matrix corresponding to variables that have not yet been fully summed but the variable must be involved in at least one of the elements currently being assembled. The assemblies can begin on one sub matrix, using further sub matrices when the first has been processed. A solution can then be obtained by back substitution.

The multi-frontal method uses the same technique as the frontal method, but instead solves for several fronts at once. To explain the concept of a front, in the frontal method, the sum can be expanded to,

$$(\ldots(((\mathbf{A}^1 + \mathbf{A}^2) + \mathbf{A}^3) + \mathbf{A}^4) + \cdots + \mathbf{A}^l)$$

Here the front can be interpreted as $(\mathbf{A}^1 + \mathbf{A}^2)$. The multi-frontal method, however, rearranges this sum so that several fronts become active at once,

$$((\mathbf{A}^1 + \mathbf{A}^2) + (\mathbf{A}^3 + \mathbf{A}^4) + \cdots + (\mathbf{A}^{l-1} + \mathbf{A}^l))$$

For more information of the multi-frontal method and its implementation the interested reader may refer to [80, 82, 83, 84, 85].

An advantage of the direct solvers described is that they can be applied to non-symmetric matrices, although the frontal method was originally devised for symmetric matrices [79]. Symmetric assumptions are also made in the multi-frontal method, although further advancements allow fast inversion of non-symmetric matrices [80].

Gaussian elimination and LU decomposition are not suitable for the scale of problems involved with the unstructured MAC method. Therefore, an algorithm based on the multi-frontal method has been chosen to solve the pressure correction equation and the implicit time stepping formulation. There exists a number of highly developed routines that apply the multi-frontal method and so to take advantage of the efficiency of these methods, the decision to use one of these as a black box has been taken. The routines used are the MA57 [83] and MA41 solvers for symmetric and non-symmetric systems provided by the Harwell Subroutine Library [86]. The efficiency of these solver rely on the quality of the basic linear algebra subroutines (BLAS) also being used. Again this is an area in which highly developed routines are available for use and so the BLAS routines from the LAPACK are used [87]. There are other implementations of this type of direct solver available, however those from the Harwell Subrotuine Library were chosen due to their availability and of their serial implemtation of the code that could easily be incorporated into the unstructured MAC method. The incorporation of the MA57 and MA41 routines into the unstructured MAC code will be discussed later.

3.9.2 Iterative Methods

The class of iterative methods is very large, where only a few will be discussed here. For a review on many method see [88, 89]. The first to be introduced, are the set of methods that include Jacobi iteration, the Gauss-Seidel method and Successive over relaxation (SOR) method, all three are very similar yet in turn produce faster convergence [77]. A commonly used alternative is the conjugate gradient method, although being slightly harder to implement it produces solutions in very few iterations, provided the problem is well conditioned (i.e. the matrix in question does not have a high condition number). In certain cases the iterations can be reduced by the use of a preconditioner.

Jacobi iteration begins by splitting the matrix $\mathbf{A}$ into its diagonal, strictly lower triangular and strictly upper triangular parts [77], so that $\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}$. The Jacobi iteration can then be expressed as,

$$\mathbf{x}^{n+1} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}^{n} + \mathbf{b}.$$

The Gauss-Seidel method gives an improvement over the Jacobi method in that the previous solution is never needed to be stored, it continually updates the solution with each position potentially using an updated value. Thus Gauss-Seidel iteration can be written as

$$\mathbf{x}^{n+1} = \mathbf{D}^{-1}\left[\mathbf{L}\mathbf{x}^{n+1} + \mathbf{U}\mathbf{x}^{n} + \mathbf{b}\right].$$

The SOR method adds a relaxation factor to the updating values added to the values from the previous time step in an attempt to speed up convergence giving,

$$\mathbf{x}^{n+1} = \omega\mathbf{D}^{-1}\left[\mathbf{L}\mathbf{x}^{n+1} + \mathbf{U}\mathbf{x}^{n} + \mathbf{b}\right] + (1-\omega)\mathbf{x}^{n}.$$

Successful implementation of both Gauss-Seidel and SOR was carried out but the convergence rate of both algorithms was deemed too slow therefore an alternative iterative method was needed.

The conjugate gradient (CG) method is one such alternative, it belongs to the class of Krylov subspace methods [89], the basic technique of which is to approximate $A^{-1}b$ by $p(A)b$, where p is a polynomial. The general idea behind the CG method is to define search directions through the space of approximated solutions, at each step converging towards the problem solution. In the CG case these search directions are a conjugation of the residuals, due to the added condition that each residual is orthogonal to the last. It is the search directions that span the Krylov subspace, hence the CG method falling into the class of Krylov Subspace methods. The derivation of the method is long and complex but leads an easy to follow algorithm that can be used without knowing how it was derived, for further information on the derivation of the CG method see [89, 90]. The resulting algorithm is then as follows,

1. Initially $r_0 = b - Ax_0$ and $p_0 = r_0$

2. Start the iteration loop, $i = 1, 2, 3....$until convergence

3. Calculate the coefficient α
$$\alpha_i = \frac{r_i^T r_i}{p_i^T A p_i}$$

4. Update the solution
$$x_{i+1} = x_i + \alpha_i p_i$$

5. Update the residual
$$r_{i+1} = r_i - \alpha_i A p_i$$

6. Check convergence based on the residuals, if converged end loop

7. Calculate the coefficient β
$$\beta_{i+1} = \frac{r_{i+1}^T r_{i+1}}{r_i^T r_i}$$

8. Update the search direction
$$p_{i+1} = r_{i+1} + \beta_{i+1} p_i$$

9. Return to top of iterative loop

The algorithm defines r_i as the residual, p_i as the search direction, x_i as the solution for an iteration i, where all these values are vectors. The coefficients α and β arise when the method is being derived [90, 89]. The conjugate gradient method requires **A** to be a symmetric positive definite matrix.

The conjugate gradient method converges fastest when the matrix is well conditioned. An ill-conditioned system, such as those produced with highly unstructured meshes, can lead to slow convergence. Faster convergence can be achieved for the conjugate gradient method by preconditioning the system. Ill conditioning can occur as an artefact of the mesh. The pressure correction equation discretisations, equation (3.14) involved the ratio between the Voronoï and Delaunay edge lengths. If these ratios vary greatly, ill conditioning can occur. An example of a mesh where greatly varying ratios could occur is when strectched mesh elements are used in part of the mesh.

A system is preconditioned by multiplying by the inverse of the preconditioning matrix **M** so that,
$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}.$$

A requirement for **M** is that the system $\mathbf{Mx} = \mathbf{b}$ is inexpensive to solve. The preconditioned system is then solved, with the slight problem with this approach being that matrix $\mathbf{M}^{-1}\mathbf{A}$ may not be symmetric or positive definite, therefore steps should be taken in the choice of **M** to ensure that symmetry still holds.

The preconditioned conjugate gradient (PCG) method can be defined as:

1. Initially $r_0 = b - Ax_0$, $z_0 = M^{-1}r_0$ and $p_0 = z_0$

2. Start the iteration loop, $i = 1, 2, 3....$until convergence

3. Calculate the coefficient α
$$\alpha_i = \frac{r_i^T z_i}{p_i^T A p_i}$$

4. Update the solution
$$x_{i+1} = x_i + \alpha_i p_i$$
5. Update the residual
$$r_{i+1} = r_i - \alpha_i A p_i$$
6. Check convergence based on the residuals, if converged end loop
7. Calculate the z array
$$z_{i+1} = M^{-1} r_{k+1}$$
8. Calculate the coefficient β
$$\beta_{i+1} = \frac{z_{i+1}^T r_{i+1}}{z_i^T r_i}$$
9. Update the search direction
$$p_{i+1} = z_{k+1} + \beta_{i+1} p_i$$
10. Return to top of iterative loop

The vector z is not needed, although the algorithm is clearer with its introduction.

Preconditioning is problem specific, a suitable pre-conditioner for a particular discretisation can be hard to find. A number of basic techniques in producing the matrix **M** are now briefly described. The simplest preconditioning technique is diagonal preconditioning, where **M** is taken to be a diagonal matrix. Its diagonal elements are the diagonal elements of **A**. A diagonal pre-conditioner is fast to produce and to invert and so its a desirable preconditioner to begin with.

Another common option is an incomplete LU (ILU) decomposition pre-conditioner [91, 89]. ILU decomposition has many levels. The technique behind ILU is to construct the matrix **M** from the lower and upper triangular matrices, such that **LU** = **M**, as in LU decomposition. These upper and lower triangular matrices are found from **A**. The technique to finding them varies slightly depending on the sparsity pattern chosen for **M**. Essentially the residual **R** = **LU** - **A** should follow the specified sparsity pattern. Once these lower and upper triangular matrices have been found $M^{-1}r_i$ may be solved by using forward and back substitution as in LU decomposition. The entire system does not need to be inverted due to the structure of the matrix M.

The easiest form of ILU to implement is ILU(0). ILU(0) retains the same sparsity pattern as **A** and so extra computational effort and storage is not required in storing a more dense matrix. L and U are found so that the elements of A-LU which are in the locations where A is zero are also zero. The standard ILU(0) algorithm as given is as follows [89],

```
1.  for i = 2,..,n: (where n is the number of rows or columns in an nxn matrix)
2.          for k = 1,...i-1:
3.                  if $a_{ik}$ not equal to zero:
4.                          compute $m_{ik}$ = $a_{ik}/a_{kk}$
5.                  for j = k+1,...,n:
6.                          if $a_{ij}$ not equal to zero:
7.                                  compute $m_{ij} = a_{ij}$ - $a_{ik}a_{kj}$
8.                  end
9.          end
10. end
```

where a_{ik} indicates the ik element in the the orginal matrix A and m_{ik} indicates the ik element of the preconditioner matrix.

Preconditioning is a wide subject area, more advanced forms include the linelet preconditioner [92]. In the case of the unstructured MAC algorithm the use of a diagonal and ILU(0) preconditioner was experimentally implemented. The results of which showed little success in improving run times and the use of preconditioners after the direct solver was implemented. The use of iterative methods and preonditioners was not investigated on stretched meshes as the direct solver gave good results. Although there is the potential that the use of a preconditioner on the stretched meshes could improve the convergence and hence run time of the conjugate gradient solver. Preconditioners have only been briefly introduced, their construction and implementation is not the focus of this work. The interested reader may refer to the literature [93, 89, 94, 95, 96] for further information on preconditioning techniques.

Chapter 4

Unstructured Co-Volume Mesh Generation

To produce stable solutions from the unstructured MAC solver, a valid unstructured mesh of the domain for the fluid flow problem in question is needed. However, there is a constraint on the type of meshes that can be used with the unstructured MAC code: they are required to be dual orthogonal i.e. the edges of the primary and dual mesh are perpendicular bisectors of each other. Using a dual orthogonal mesh allows the scheme to retain its second order accuracy. These dual orthogonal meshes shall be known as co–volume meshes.

This chapter introduces the requirements for a co–volume mesh and discusses how suitable meshes can be produced. There are many methods for generating unstructured meshes, two of which are the Delaunay triangulation and the advancing front method. There exist highly developed, efficient mesh generators that use these methods and so it is desirable to be able to generate meshes using these procedures that are suitable for use with the unstructured MAC method. With this in mind, a number of techniques that modify a general unstructured meshes, to produced a suitable co-volume technique have been tested. This ranges from merging elements to quadrilaterals, moving the element centres or optimising the whole mesh to find suitable node and element centre locations [72]. Another approach is to use a method that generate a suitable mesh without the need for modification, such as the stitching method [24]. The stitching method still makes use of the advancing front technique. Due to the necessity to use the Delaunay triangulation mesh generation technique and the advancing front mesh generation technique, a brief description of both is given.

The structure of this chapter is as follows, initially the requirements for a co–volume mesh are detailed. Following this a brief description of general unstructured mesh generation techniques, including Delaunay triangulation and the advancing front method will be given. Then four methods for creating the required co–volume meshes are detailed. These include: mesh merging; element centre moving; optimised meshes and the stitching method.

4.1 Co-Volume Mesh Requirements

To produce second order accuracy in the unstructured marker and cell scheme, a dual orthogonal mesh must be used. In the unstructured framework, the Delaunay-Voronoï diagram provides the means to fulfil this criteria. To fully satisfy the dual orthogonal condition, each Voronoï and Delaunay edge must be perpendicular bisectors of each other. The mesh that completely meets this criteria is known as the ideal mesh.

Using the Delaunay-Voronoï framework, the ideal mesh is a triangulation formed of equilateral triangles. In this case, the number of nodes connected to each node is six, hence producing Voronoï cells with six sides. Each Voronoï edge length $h^v \approx 0.56h^d$, where h^d is the Delaunay triangle edge length and all triangle angles are 60^o, as defined in [24].

An ideal mesh is not suitable for complex geometries and so the dual orthogonal criteria must be relaxed. It is therefore, necessary to define the requirements to be enforced on the Delaunay-Voronoï dual mesh as follows [24],

1. Each Voronoï vertex must lie inside the corresponding Delaunay triangle.
2. Any deviation of the angle between the Delaunay edge and its intersection Voronoï edge from 90^0 should be kept to a minimum.
3. The deviation in location of this vertex from the Delaunay triangle circumcentre should be minimised.
4. Any deviation in the location of the midpoint of a Voronoï edge from the point of intersection with the Delaunay edge should be minimised.
5. Any deviation in the location of the midpoint of a Delaunay edge from the point of intersection with the Voronoï edge should be minimised.

Producing meshes that conform to these requirements will ensure that the mesh is suitable for successful simulation using a co-volume method. The first criteria can be considered to be the most important. When meshes do not comply and a Voronoï vertex is located outside the corresponding Delaunay triangle, instabilities can occur in the unstructured MAC method. In producing meshes that conform this first criteria, it is often necessary to relax the other four criteria, but as mentioned any deviations should be kept to a minimum.

As the first criteria is most problematic an element that does not conform to it shall be known as a bad element. The type of triangular element that causes this problem are obtuse triangles. An obtuse triangle means its circumcentre, the Voronoï vertex, lies outside the triangle. An example of a bad element is shown in figure (4.1).

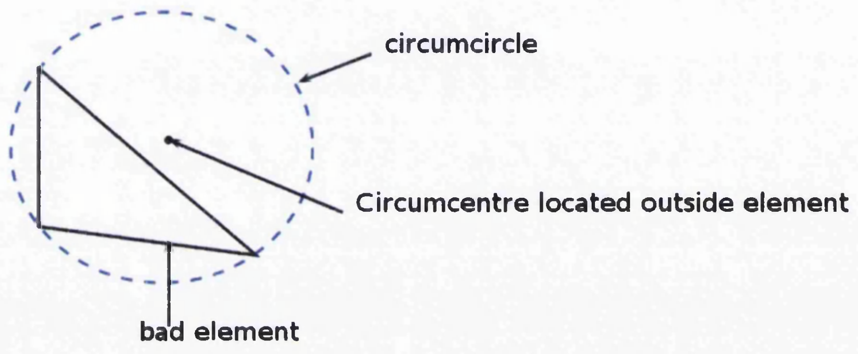


Figure 4.1: Example of a "bad element"

The traditional automatic mesh generation methods of advancing front and Delaunay triangulation are not designed to create meshes meeting the requirements. Although in some cases meshes that conform well enough to these criteria can be produced, it is not the general case. The difficulty in producing suitable meshes using advancing front or Delaunay triangulation increases as the required geometry becomes more complex. Therefore, methods which can be used to improve the quality of these general meshes have been devised. The first deals with the problem of short Voronoï edges, through a technique known as mesh merging. Mesh merging is a technique applied to the mesh within the unstructured MAC code, it is therefore, a technique that may be applied to any mesh used with the unstructured MAC algorithm. The second technique is again implemented within the MAC code and helps to deal with bad elements. This is achieved my moving the location of the element centre of the bad elements, by using the primitive method of averaging a Delaunay triangles circumcentre with its barycentre. The third technique expands the approach of moving the location of the element centre, using a mesh optimisation method. It is a technique applied to a general mesh prior to use with the unstructured MAC algorithm. The fourth technique, known as the stitching method, is a mesh generation procedure. The stitching method generates meshes that are primarily formed of an ideal mesh and so the mesh should not contain any bad elements.

4.2 General Unstructured Mesh Generation Techniques

The majority of meshes used with the unstructured MAC method began from mesh generators that use either Delaunay triangulation or the advancing front method, although they may have been modified afterwards. This section briefly describes some common processes of mesh generation and the basic premise behind the two mentioned methods.

4.2.1 Basic Mesh Generation Components

There are some features that are common in the construction of all unstructured meshes [29, 23, 17, 22]. All mesh generation methods should include an automatic node generation procedure, an important part of which is to ensure the boundary node distribution is sufficiently smooth.

The density of the automatically generated nodes can be controlled in two ways, using a point density or a grid source. The point density describes the density of the nodes over the domain with higher densities producing more nodes in that region. It is usually controlled by a background mesh. This is a coarse mesh that specifies a density at each of its nodes, any actual mesh nodes automatically created will apply the density specified by the background mesh in the region they are created in. For more complex meshes, grid sources may be required. A grid source works with the grid point density but it defines a different density around a specified point or line. The purpose of using grid point and source densities is to allow a mesh to be easily refined in any region that refinement might be required, for example in a wake region.

4.2.2 Delaunay Triangulation

The Delaunay triangulation procedure that follows, summarises the work of Weatherill [17]. The basic idea of which is to apply the Delaunay triangle criterion. The Delaunay triangulation criterion can be defined from the definition of Voronoï regions. For a set of points $\{p_i\}$, which can be interpreted as the nodes in a mesh, the Voronoï region $\{V_i\}$ can be defined as,

$$\{V_i\} = \{p : ||p - p_i|| < ||p - p_j||, \forall j \neq i\}.$$

the Voronoï region $\{V_i\}$ is the set of all points p that are closer to p_i than any other point. Therefore, in two dimensions the boundary which will form a side of a Voronoï polygon is midway between the two points it separates. This boundary is a segment of the perpendicular bisector of the straight line that joins the two points. Perpendicular bisectors are used to connect the nodes allowing a Delaunay triangulation to be formed. By definition, these lines are orthogonal to each other.

A property of the Delaunay triangulation that allows its successful use for mesh generation is the in-circle criterion. The in-circle states that for a triangulation $T(p_i)$, no point of the set p_i is interior to the circumcentre of any triangle of $T(p_i)$, see figures (4.2,4.3). An arbitrary triangulation can be transformed into a Delaunay triangulation by repeated application of the in-circle criterion.

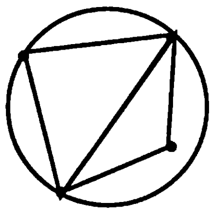


Figure 4.2: In-circle criterion not satisfied for four points

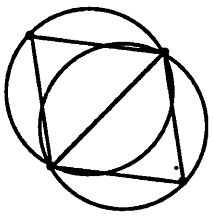


Figure 4.3: Four points that satisfy the in-circle criterion

Delaunay mesh generation begins with the nodes being defined on the boundaries. A Delauany triangulation is then produced using the criteria defined above. Nodes are then added at the centroids are each element and the elements are sub divided following the triangulation definitions [17, 72].

Delaunay triangulation is a good option for producing a co–volume mesh, as all meshes produced by the Delaunay method have each Voronoï edge perpendicular to the Delaunay. However, this property may be lost in two ways one being due to the boundary discretisation, the other because the vertices of the Voronoï cells are forced to be located at the triangle circumcentres, hence, producing bad elements.

4.2.3 Advancing Front

The advancing front method for unstructured grid generation as described in [29, 23] starts by taking the boundary discretisation and creates new nodes inside the domain to create elements. The process advances until the whole domain is filled. Regions can be predefined to have a certain mesh spacing, the generator will then work with these values. This is achieved via the grid point density background mesh.

In order to construct the mesh, the sequence of straight line segments that connect the boundary nodes are stored in an array called the generation front. At any time in the mesh generation process, the generation front contains all possible sides that could be used to make a new triangular element. Sides that have already been used to create a triangle are removed from the front, hence, all that is stored is the boundary between existing elements and empty space. The algorithm for two-dimensional mesh generation as given in [23] is described as follows:

1. Select a side from the generation front, using a selection criteria.
2. Calculate the centre of the selected side.
3. Determine the ideal position of the new node. This lies on the perpendicular bisector of the selected edge.
4. Make a list of all possible node locations. These include all generated nodes that could provide the ideal triangle.
5. The new node is selected as the one that produces the best triangle.
6. The new triangle is stored and the generation front updated.

Applying this algorithm provides an efficient way of producing meshes for any domain.

4.2.4 Hybrid Meshes

For certain types of flow, solver performance can be improved by the use of a hybrid mesh. Hybrid meshes usually incorporate stretched rectangular elements into a boundary layer along boundaries of the mesh. The purpose of doing so is in the simulation of highly viscous flows. On the boundaries of objects in such flows, detail will need to be captured in the normal direction out from the object. The use of stretched rectangular elements allows mesh refinement in the normal direction but not the tangential. Hence, reducing the number of overall elements and thus reducing the required computation time.

Hybrid meshes are created using the advancing layer technique by introducing a stretching parameter. The stretching parameter can be gradually increased to one at a suitable distance from the boundary, depending on the number of boundary layers that are required [29]. The stretched boundary layer elements are often produced as nearly right angle triangles and so they can be merged to a quadrilateral, hence a hybrid mesh is formed.

4.3 Mesh Merging

Mesh merging aims to improve the quality of a mesh by forming a quadrilateral from two triangular elements. Triangles should be merged when a short Voronoï edge is found in the mesh. Short Voronoï edges do not necessarily mean the co–volume mesh criteria is not met but they restrict the size of the time step and so removing them is beneficial, the reason for which will now be discussed.

Small Voronoï edges occur when triangles have an angle that is much greater than 60^o yet smaller than 90^o. Two neighbouring right angled triangles pose another problem of a zero Voronoï edge. These short or zero edges occur because the Voronoï vertices are located at the Delaunay triangle circumcentres. As the angle opposite to the shared side of two neighbouring triangles becomes closer to 90^o, the circumcentre moves closer towards the shared edge. The main problem with these short edges is they reduce the size of the time steps as can be seen in the equations (3.25,3.26). A small time step greatly reduces the efficiency of the algorithm. The solution to the short Voronoï edge problem, is to merge the two triangular elements that form the short Voronoï edge to a quadrilateral, see figures (4.4,4.5,4.6). The merging process removes a Delaunay edge

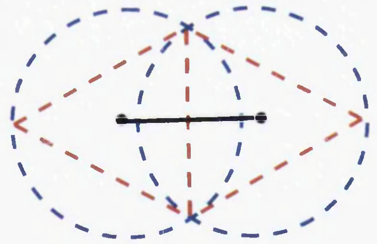


Figure 4.4: Equilateral triangle, where case no merging needs to occur.

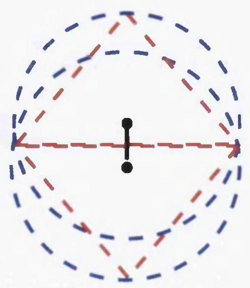


Figure 4.5: Triangles with one angle approaching 90^o, producing a short Voronoï edge.

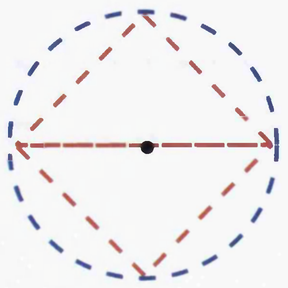


Figure 4.6: Right angle triangles, producing a zero Voronoï edge.

and the problematic Voronoï edge from the domain. To determine whether a Voronoï edge should be removed, a merging factor is specified. If the Voronoï edge is shorter than the Delaunay edge multiplied by this factor then the edge should be removed [24]. This approach makes no physical changes to the mesh, therefore, there is no need to produce new meshes which employ this technique in the generation stage. An example of where merged elements may occur is displayed in figure (4.7).

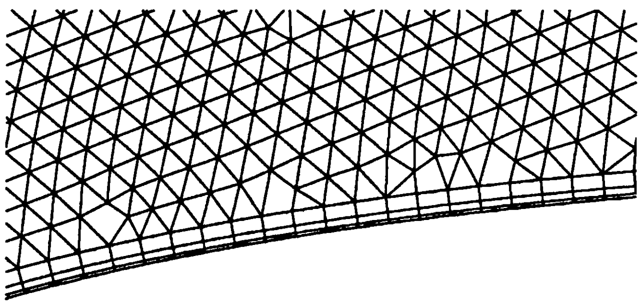


Figure 4.7: Example of merged elements

Mesh merging by the criteria laid out above, is beneficial to the stability of the algorithm as it helps to retain orthogonality. To further explain this property, if two neighbouring triangles are right angled, a quadrilateral with all interior angles equal to 90^o is produced. If all surrounding elements to the quadrilateral are equilateral or right angled triangles then orthogonality will be retained. The merging factor will allow triangles with angles close to right angles to be merged. Only some loss in orthogonality will occur, not enough to be detrimental to the accuracy of the solution.

Merging to a quadrilateral presents no problem to the discretisations. Although they have been given in the terms of a triangular mesh, they also apply to quadrilateral mesh elements. Therefore, merging elements does not require a change to the discretisations. In fact the momentum equation discretisations can apply to mesh elements with any number of sides. The only challenge restricting the limit is the reconstruction of the tangential velocity, a process that is currently limited to triangular or quadrilateral elements, more on this subject will be mention later.

4.4 Cell Centre Moving

A simple yet effective solution to the bad element issue is to relax the condition that the Voronoï vertex must be located at the corresponding Delaunay triangle circumcentre. Of course when a bad element occurs, the requirement that the Voronoï vertex is located inside the Delaunay triangle is violated. This is the more important condition as a vertex outside the triangle completely removes any orthogonality of the meshes, as the edge connecting the Voronoï verticies no longer intersects the Delaunay edge.

The simple fix is to move the vertex, no longer locating it at the circumcentre but at some other point. Initial experiments moved the Voronoï vertex to the centroid of the Delaunay triangular, this however removes orthogonality to an extent that reduces accuracy too much. An approach that does work is to take an average of the circumcentre and the barycentre, a crude fix that produces satisfactory results. This is possibly due to the vertex then being located close to one of the Delaunay edges, a situation that can produce short Voronoï edges and hence cells are merged. The new quadrilateral locates the new Voronoï vertex at the midpoint location of the previous two Voronoï vertices that would have existed. This approach helps retain orthogonality on all the quadrilateral edges. This is visualised by looking at the Voronoï mesh when the various vertex locations are used. Figure (4.8) shows the Voronoï mesh when the circumcentre is used, it can be seen that in some cases the direction of some Voronoï edges are inverted. Figure (4.9) shows the Voronoï mesh when its vertices are located at the Delaunay triangle centroid, where it can be seen that there is a loss in the dual orthogonality requirement for several elements. Figure (4.10) shows the case when the average of the two is used.

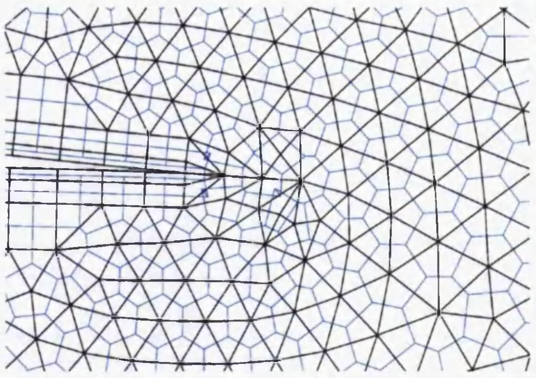


Figure 4.8: Voronoï mesh using circumcentre

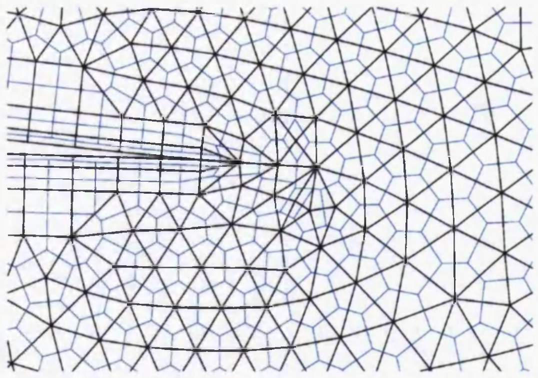


Figure 4.9: Voronoï mesh using centroid

It can be seen that similar effects to using the circumcentre occur, but their severity is greatly reduced and thus there are short Voronoï edges. This allows mesh merging to take place.

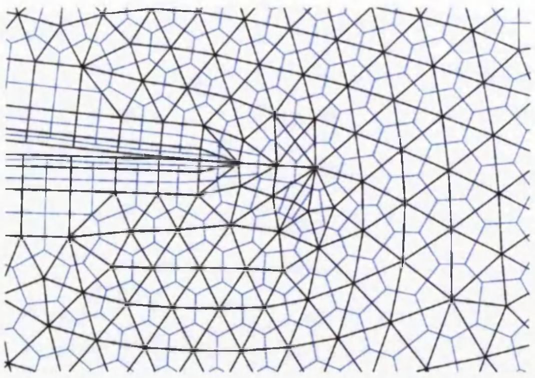


Figure 4.10: Voronoï mesh using averaged centre

4.5 Optimised Meshes

Mesh optimisation deals with the problem of removing bad elements from the mesh and keeping deviations of the Voronoï vertex from the Delaunay element centres and the deviation of the point of intersection of the both edge midpoints to a minimum.

The problem with moving the cell circumcentre is that it fixes the bad element problem but does not ensure that the mesh globally conforms to the other criteria. The bad element can be removed but the afore mentioned deviations are not kept to a minimum. The orthogonality requirement may also be lost as edges are no longer perpendicular to each other. A solution to this problem is to optimise the position of the Voronoï vertex globally. The method attempts to ensure that all edges remain perpendicular. In order to make this possible, the requirement that the intersection be at the Delaunay and Voronoï edge midpoints is relaxed.

To explain this process it is convenient to look at the position of the circumcentre from the intersect of all the perpendicular bisectors. Circles of the equal radius centre on each triangle verticies cannot be used to find a interest of all perpendicular bisectors that lies inside the triangle. This intersect is the location of the Voronoï vertex, see figure (4.11). An idea that has been experimented with is that of changing the Voronoï vertex positions by employing a weighted Voronoï power diagram [97, 98]. The weighted Voronoï diagram essentially assigns a weight to each of the Delaunay vertices, to construct the circumcentre these weights are all equal. However, to move the Voronoï vertex away from the circumcentre, these weights vary. This has the effect of changing the radii of the associated circles and thus moving the intersect of the constructed bisectors, see figure (4.12). The value of these weights can then be optimised to find the optimal combination and thus the best possible mesh. A desirable trait of

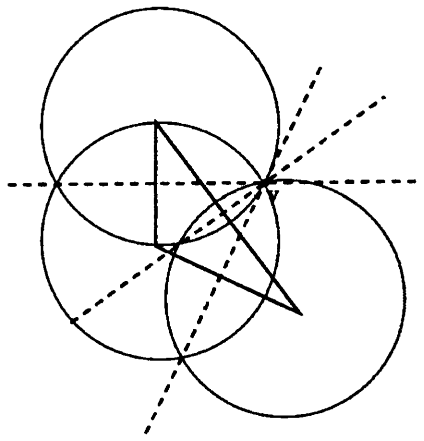


Figure 4.11: Cell centre located outside the triangular element using a equal radius construction

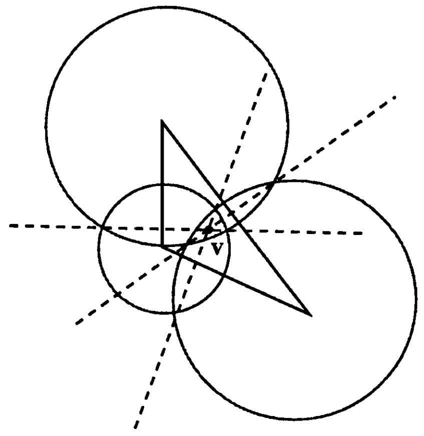


Figure 4.12: Cell centre located inside the triangular element using a weighted radius construction

the weighted Voronoï method is that it retains orthogonality. The Voronoï edges remain perpendicular to the corresponding Delaunay edges. However, the orthogonality is not dual. The Voronoï edges no longer bisect the Delaunay edge and so the intersection point is no longer located at the midpoint of the edges. This means that any central differencing techniques used in the discretisation are first order accurate rather than second. Due to the variable locations, there are not enough point information to use a higher order central difference method to retain the second order accuracy [10]. The sacrifice is a necessary one and suitably accurate results are still obtained using these methods.

A recent application of the weighted Voronoï approach combines its usage with that of an optimisation process of the Delaunay vertex locations in an attempt to produce the best mesh possible. The Delaunay vertices are first optimised using the modified cuckoo search algorithm, a gradient free optimisation technique [99, 72, 100]. As this process on its own does not remove all bad elements, it is then combined with the Voronoï power diagram. A further optimisation of the position of the Voronoï vertices is performed using a combination of the modified cuckoo method and proper orthogonal decomposition. Proper orthogonal decomposition is a dimension reduction technique, the meshing optimisation problem has a very high number of degrees of freedom which needs reducing to a few hundred to allow optimisation to be applied. Once optimisation is applied to a mesh, the position of Delaunay and Voronoï vertices are moved, producing a mesh that no longer contains bad elements, see figures (4.13,4.14). It is

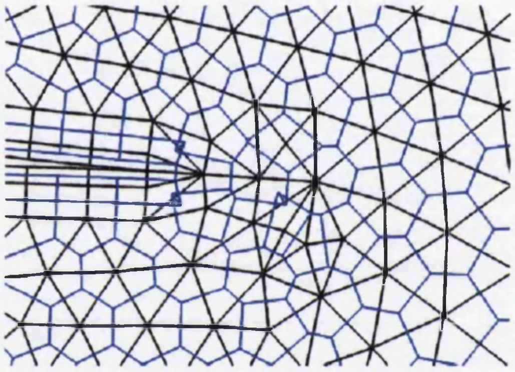


Figure 4.13: Mesh with bad elements

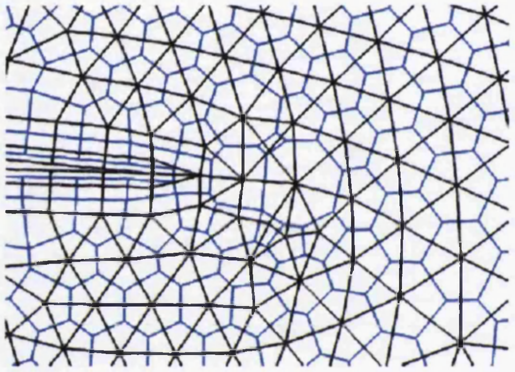


Figure 4.14: Mesh apart optimisation

this optimisation process that is applied to the general meshes when they fail to produce stable or accurate solutions.

It should be noted that the mesh optimisation technique have only been developed for meshes with triangular elements and therefore is not applicable to hybrid meshes. It is also not applicable to stretched triangular meshes, the optimisation process causing problems with the stretched triangular boundary elements, see figure (4.15). Like all the meshing codes, the mesh optimisation routine has been developed by another author and meshes have been produced for the author of this thesis for use with the unstructured MAC method. Therefore, no improvements to overcome this issue within the mesh optimisation algorithm have been implemented by this author. However, a simple fix has been implemented, by manipulating the meshes before their input to the optimiser. The manipulation is simple in that the quadrilateral elements are split from the triangular elements, the triangular element information can then be sent to the mesh optimiser. After the triangular elements are optimised they must be rejoined with the

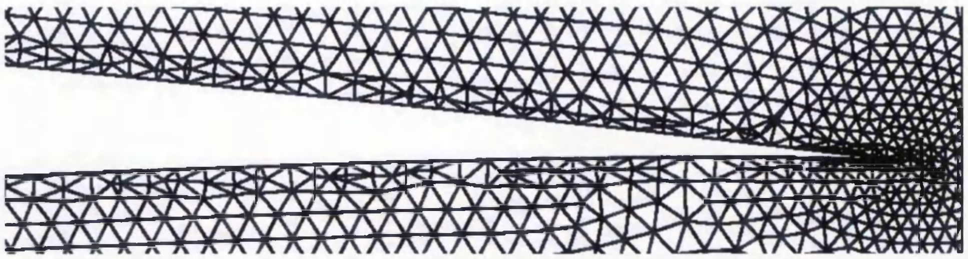


Figure 4.15: Unsuccessful mesh optimisation on stretched boundary elements

quadrilateral element data. An algorithm was coded to carry out these tasks. A slight problem with this approach, is that it can produce a slight loss in orthogonality. This is because a triangle circumcentre may not align with the quadrilateral one after they have been moved during optimisation. The effects of this have been observed to be minor, the cost of which outweighs the benefits. In some cases, a stable solution is not possible without this optimisation process. Examples demonstrating these cases are presented in the results chapter.

4.6 The Stitching Method

The ideal mesh as previously defined will not fit general boundaries. This is the same problem as with a square Cartesian mesh, thus if the unstructured MAC method were restricted to using the ideal mesh, it would offer no benefit over using the original Cartesian method. The ideal mesh is therefore not fit for purpose of modelling flow around complex geometries, the purpose of the unstructured MAC method.

The stitching algorithm presents an alternative approach to those already mentioned. It defines a mesh generation technique, producing suitable meshes that do not need modification. The technique was originally developed for use with the Yee algorithm, a co-volume scheme for electromagnetics [24]. The stitching algorithm uses a modified advancing front method to produce a local high quality mesh that fit the boundary. This is then stitched to the ideal mesh to fill the remainder of the domain. This method ensures a good overall mesh quality, however, in the stitched region some elements with short Voronoï edges can occur. A problem that is simply fixed by the merging technique.

The stitching method should allow the creation of meshes that do not have any bad elements. Although the likely place they would occur is in the advancing front mesh generation stage and the region in which this is stitched to the ideal mesh. For meshes fitted to aerofoils where the high quality mesh is required in boundary region, the problem of bad elements could still yield unstable solutions. A limitation of the stitching method is that the mesh resolution must remain constant, there is no option to refined the mesh in certain regions. Therefore, if a refined region is required in a mesh, such as wake region, then the entire mesh must be of that resolution. Finer meshes require more computational effort and a refined mesh in regions that it is not required, creates unnecessary over heads.

Chapter 5

Code Implementation

Implementation of the unstructured MAC algorithm, into a modular Fortran 90 code, can be split into three parts. The initial part requires knowledge of the domain mesh, which will vary for each given flow problem. This mesh needs to be processed and manipulated to form data structures suitable for use with the unstructured MAC discretisations.

Following the processing of the mesh information, the discrete Navier-Stokes equations can be solved using the unstructured MAC algorithm, as given in Chapter 3. This solution step will include the implementation of a Poisson equation solver. The last part in the code implementation requires the results to be processed into a suitable format that can then be output for visualisation.

This chapter aims to describe the implementation of each of the three parts. The first section discusses the predefined information required by the unstructured MAC code. The following section details the unstructured MAC code format and all the data structures that are required. This section gives further detail on the more complex processes contained within the unstructured MAC algorithm. Finally the type of information output form the method and how they are calculated is discussed.

5.1 Predefined Input

In defining a fluid flow problem, there are several pieces of information the unstructured MAC algorithm requires. The most significant piece of information is the discrete representation of the domain, i.e. the mesh of the domain. In the case of this project the mesh is produced using in-house mesh generation software and so the unstructured MAC code has been tailored to process this format. There is no reason why any other mesh generation software cannot be used provided the following pieces of information are defined:

- The element connectivities, i.e. The three nodes that form each Delaunay element. In the case of quadrilateral elements, each element will have four nodes. The unstructured MAC code is capable of processing both triangular and quadrilateral elements, or a combination of both.
- The Cartesian coordinates of all the nodes.
- The boundary edge definitions and the boundary condition.
- An optional item of the Cartesian coordinates of the element centres may also be given in the case where an optimised mesh is being used.

In the next section the processing of the given mesh information into a suitable format for the unstructured MAC algorithm is described.

As well as the definition of a mesh, there are two other pieces of predefined information, the first specifies all the information required to model a particular fluid flow problem. Included in this information must be the mesh file name, the output file names, any boundary velocities given in Cartesian components and finally the Reynolds number. The initial conditions for the domain as taken to be the inflow velocity.

The final set of information required for successful simulation are all the variable information that configures the MAC solver. This includes information on whether an optimised mesh is being used, in this case extra information is provided in the mesh file and element centres do not need to be calculated. The mesh merging factor must also be defined, the mesh merging factor governs whether a Vornoï edge is too short and should be removed from the domain. The choice of merging factor is determined by a trail and error approach. When more complex geometries are implemented, the quality of the solution can be very dependent on this factor and so the best possible factor that does not try and merge elements to shapes with more than four sides has to be determined.

Another notable piece of predefined information is the stopping criteria. For steady state problems this is some convergence criteria and as convergence to five orders of magnitude has been deemed a suitable level, this has been built into the code.

The MAC algorithm is also capable of solving unsteady problems. Unlike steady problems which will stop after the convergence criteria is met, unsteady problems need some other form of stopping criteria to be defined. For the MAC algorithm this is a specified number of time iterations.

5.2 Solver Implementation

In producing source code for the MAC algorithm, initially there needs to be a preprocessing stage to interpret all the mesh information. As the format being used for the mesh file is the result of an in-house mesh generator there exist some in-house source code for manipulating the data. The in-house source code contains algorithms for constructing the mesh information, calculating the element centres and volumes and for merging mesh elements. After the preprocessing stage, the discretisations and MAC algorithm given in chapter 3 can be implemented into the MAC algorithm source code.

The final version of the source code can be split into three sections: mesh manipulation, the MAC solver and direct solvers for implicit systems. The direct solver may not be required in all cases. As the source code is implemented in Fortran 90, each section can be placed in a module. The purpose of each of these modules and the data structures they require will now be described.

5.2.1 Mesh Manipulation Data Structures

A major stage in the code implementation is how the mesh information is interpreted. During mesh interpretation, all data structures can be defined. Values can be assigned to many of these data structurs directly from the mesh information. Others however, need some manipulation in order to assign their values.

The mesh manipulation module defines a type containing all the information that defines the mesh or can be derived from it. The data structures contained in this type are as follows:

Mesh size variables: A set of integer values storing the number of elements, nodes, boundary edges and internal edges.

Connectivities: Integer array of dimension number of elements by three, storing the nodes that form each triangular element. A visual representation of the array is now displayed,

$n1_1$	$n1_2$	$n1_3$	..	..	..	$n1_{ne}$
$n2_1$	$n2_2$	$n2_3$	..	..	..	$n2_{ne}$
$n3_1$	$n3_2$	$n3_3$	..	..	..	$n3_{ne}$

where $n1, n2, n3$ refer to the three nodes and the number in the subscript refers to the element number, which ranges from one to ne, the total number of elements.

Forced merging array: Integer array of length corresponding to the number of elements. These values are only non zero if quadrilateral elements appear in the mesh data file. A quadrilateral element is interpreted as two triangular which are later merging. When this situation occurs, the element that a current element should be merged to, to form the quadrilateral is stored.

Coordinates of nodes: A two dimensional real array, the length of which is equal to the number of elements. The x and y coordinates of all mesh nodes are read from the mesh data file and stored. In the visual representation,

x_1	x_2	x_3	..	..	..	x_{np}
y_1	y_2	y_3	..	..	..	y_{np}

the subscript number refers to the node number, which ranges from one to np, the number of nodes.

Coordinates of element centres: A two dimensional real array, the length of which is the number of nodes and width two. The x and y coordinates of triangular element centres are stored. If mesh optimisation has been used, these will be predefined, if not, they can be calculated from the connectivities. Again the array can be visualised in a similar fashion to the connectivities,

x_1	x_2	x_3	..	..	..	x_{ne}
y_1	y_2	y_3	..	..	..	y_{ne}.

Edge information: This is an integer array of dimension that corresponds to the number of sides. All information needed to define an edge in the mesh is stored. Initially the boundary edges are read from the mesh data file. Other sides are then constructed from the mesh information.

It is not enough to define an edge by just its too connecting nodes, both elements that lie on either side of edge should also be included in the definition. Therefore, the required data structure stores both the connecting nodes $n1$ and $n2$ and the neighbouring elements $e1$ and $e2$, that is every edge in the mesh. In the array visualisation,

$n1_1$	$n1_2$	$n1_3$	..	..	..	$n1_{ns}$
$n2_1$	$n2_2$	$n2_3$	..	..	..	$n2_{ns}$
$e1_1$	$e1_2$	$e1_3$	..	..	..	$e1_{ns}$
$e2_1$	$e2_2$	$e2_3$	..	..	..	$e2_{ns}$

the subscript refers to an edge in the domain. This value ranges from one up to the number of sides, ns.

Length of the Delaunay triangle edges: Real array which has a length equal to the number of edges that stores the Delaunay triangle edge lengths. Lengths are calculated from the node coordinates and so require the edge information.

Length of the Voronoï element edges: Each Voronoï edge is identified by the Delaunay edge it intersects. Therefore, a real array which has a length equal to the number of edges is required to store the Voronoï element edge lengths. Lengths are calculated from the element centre coordinates and so the edge information is required. A Voronoï side is given the same numbering as the Delaunay side it intersects.

Area of the Delaunay elements: Real array that stores the area of each Delaunay element. This can be calculated from the connectivities and node coordinates.

Area of the Voronoï elements: Real array that stores all the areas of the Voronoï elements. The centre of each Voronoï element are the mesh nodes and so each Voronoï element can be identified by each node number. The area can be constructed as the sum of the areas of smaller triangles that make up the Voronoï element, these can be found by using the edge lengths.

Edge Tangents: Real array, storing both vector components τ_1 and τ_2 for the tangent to each edge. The tangents are calculated from the side information. If an edge is connected by the two nodes $n1$ and $n2$, the convention taken in calculating the tangents is $\tau_1 = n1_x - n2_x$ and $\tau_2 = n1_y - n2_y$, where the subscripts x and y refer to the x or y coordinate. The visual representation, where the subscript refers to the edge number is as follows,

$(n1_x - n2_x)_1$	$(n1_x - n2_x)_2$	$\dots$	$(n1_x - n2_x)_{ns}$
$(n1_y - n2_y)_1$	$(n1_y - n2_y)_2$	$\dots$	$(n1_y - n2_y)_{ns}$

Edge Normals: Real array, storing both vector components for the normal to each edge. The normal to an edge is perpendicular to the tangent and so it is calculated as the right hand skew of the tangential vector components. This can be visualised as follows,

$(n1_y - n2_y)_1$	$(n1_y - n2_y)_2$	$\dots$	$(n1_y - n2_y)_{ns}$
$(n2_x - n1_x)_1$	$(n2_x - n1_x)_2$	$\dots$	$(n2_x - n1_x)_{ns}$

Both the tangent and normal values do not represent the unit tangent and normal and so will need to be divided by the magnitude.

Edges that form an Delaunay element: Integer array, that stores all the edges that create Delaunay elements. Elements may be triangular or quadrilateral and so the array allows up to four edges to be stored for each element.

Normal directions for each element: Real array of the same dimensions as the previous array. The discretisation of the convective term requires the edge normal to be an outward normal. If it is not, then the velocity component used in the calculation will have the opposite sign to what it should. As these signs do not change throughout the computations, they are stored for efficiency. The value stored is the dot product between the edge normal and the element outward normal, which is either a one or a minus one.

Edges that form a Voronoï element: Integer array that stores all edges used to create a Voronoï element. Depending on the structure of the mesh, the number of sides around a Voronoï element can vary, unlike a Delaunay element the number of edges is not restricted. All edges around a particular element are grouped together and stored in a long array. For example,

$$\underbrace{\boxed{1}\boxed{5}\boxed{10}\boxed{12}\boxed{14}}_{\text{Sides around 1st Voronoï element}}\underbrace{\boxed{12}\boxed{6}\boxed{21}\boxed{10}\boxed{9}}_{\text{Sides around 2nd Voronoï element}}\;.....\;\underbrace{\boxed{41}\boxed{69}\boxed{99}\boxed{32}\boxed{55}}_{\text{Sides around nth Voronoï element}}$$

Using this format requires two look up arrays to identify the position of the edges for a particular Voronoï element. The first of these arrays stores the number of edges each Voronoï edge has. The second stores the initial location of the first edge for each Voronoï element. Knowledge of all the edges around each Voronoï element is only required when using an implicit time stepping routine.

Tangent directions for the Voronoï element: Real array that stores the direction of the tangents around each Voronoï element. The format of the array is the same as that above and requires the use of the location and number of sides arrays. The viscous term discretisation relies on an anti-clockwise tangent around the Voronoï elements. Similarly to the convective term this direction is predefined as a clockwise tangent an opposite sign will be present in the discretisation. Therefore, the dot product between the Voronoï edge tangent and the anti-clockwise tangent needs to be calculated to give the correct sign. This value only need be calculated if using an implicit time stepping routine, when using an explicit scheme it is acceptable to calculate the sign when it is required.

The subroutines contained in the mesh manipulation module are sufficient to manipulate all the data supplied in the mesh data file into a form suitable for use with the unstructured MAC discretisations. Mesh merging and mesh quality checking algorithms are also contained within this module, a brief discussion on their purpose and implementation will now be presented.

Mesh merging is required when short Voronoï edges exists, as discussed in the previous chapter. These short edges are identified using the merging factor and then flagged for removal. Once an edge has been removed the two elements are reconstructed as one, therefore, a new element centre and volume needs to be assigned. For the new element centre, the average of the two original triangular cell centres is taken. Whereas, the volume is the sum of the original element volumes.

Quadrilateral elements that have been interpreted as two triangles are also flagged for merging, they are not directly read into the code. This is due to the algorithm that allocates edge information, being an in house algorithm which has been designed for interpreting triangular elements. Since merging may occur anyway, there are no excessive overheads in first interpreting them as two triangles and later merging to the original quadrilateral. The splitting then merging approach is therefore necessary to avoid modification to the already designed algorithm. The two triangular elements to be merged form the original quadrilateral are identified at the mesh read stage. The merging criteria is extended to include these elements, therefore, flagging the joining edge for removal. When these two triangles are merged back together, the centre of the element is recalculated as the centre of mass of the quadrilateral.

After merging has taken place, the edge information array, element volume arrays and element centre arrays are reordered. Edge lengths for the new ordering can then be calculated. Several of the defined data structures are assigned by manipulating the edge information. The calculation these values should be reserved after the mesh merging stage, so not to unnecessarily recalculate information.

The quality of a mesh can be analysed by finding the number of bad elements contained in the mesh. Prior to mesh merging, a bad triangular element can be identified as an obtuse triangle, therefore any triangle that has an interior angle greater than 90^o can be flagged as an bad element. The number of identified bad element is used as a diagnostic tool, it serves as an indicator for a potential cause of any instability. In practice, it is not only the number of bad elements but also the position of them that effects the stability of a solution, this will be demonstrated in the results chapter.

Identifying the bad elements by definition, identifies that the circumcentre lies outside the triangle. Therefore, it is at this stage that the change of the element centre from the circumcentre to an average of the circumcentre and barycentre can be implemented.

5.2.2 MAC algorithm data structures

The mesh information has been manipulated, to the correct format allow the discrete form of the Navier-Stokes equations to be solved efficiently. The next module in the MAC algorithm source code defines a data type to store all information relating the the solution of the Navier-Stokes equations. The routines defined in this module allow initialisation and solution of a given fluid flow problem. The data structures required in the MAC data type are defines as:

The time step: Real array, storing the time step for each edge. If a global time step is required, the minimum is found and assigned to every position in the array. In the explicit scheme the time step is recalculated at each iteration.

Normal velocity variables: Real arrays that store the initial and new normal velocity components for every edge in the domain.

Tangential velocity component: Real array, stores the tangential velocity for each edge, as constructed form the normal velocities stored.

Pressure: Real array, stores the pressure at each Delaunay element centre.

Right hand side of the pressure correction equation: Real array, stores the correction value ψ for each Delaunay element.

Curl: Real array, stores the curl of the velocity around Voronoï elements, the centres of which coincide with each node.

Using this data structure, the MAC algorithm can be followed to give a solution to the Navier-Stokes equations. The various stages of the implemented MAC source code include: the initialisation, time step determination, solution of the intermediate velocity, solution of the pressure correction equation, correction of variables, reconstruction of the tangential velocity and output of the results.

Several of these processes are implemented by following the discretisations described in chapter 3. However, some extra information can be given on a few of the processes.

The solution of the pressure correction equation requires an implicit solver. This can either be the conjugate gradient method or a direct solver, the implementation of which is detailed in the next section. Regardless of the solver being used, the sparse matrix produced in the implicit system formulation needs to be initialised and allocated. Similarly when using an implicit time stepping routine a sparse matrix needs to be initialised and allocated. For a sparse matrix only the non zero values are stored and so the location of these non zero values must also be stored.

At the initialisation stage, the pressure is initially set to zero over the entire domain. Whereas, the velocity is set to the inflow or moving wall value to prevent a shock wave developing in the domain.

When determining the time step size, an adaptive time stepping routine can is implemented within the explicit scheme. The time step size is be calculated at the beginning of each time iteration loop before the Navier-Stokes equations are solved. When using semi-implicit time stepping it is more beneficial to use a fixed time step. The viscous term requires knowledge of the time step and so the implicit matrix would need formulating every iteration if it were to change. This is a costly process and so the benefits in using the smallest time step possible each iteration are negated by having to recalculated these values.

Implementation of the viscous effects on boundaries also requires some thought when implementing the semi-implicit time scheme. Voronoï elements that contain a boundary edge require knowledge of the tangential velocity, which may change every time step. The tangential velocity from the previous time step can be taken and so this can be deemed as a known quantity, thus placing it on the right hand side of the implicit system. In most cases the tangential velocity along a boundary can be taken to be zero and so the extra computations in assigning these values are not required. However, one case where it is needed, is when a moving wall boundary has a defined value for only the tangential velocity. The normal velocity is zero in this case and subsequent tangential velocities will not be recovered if all the normal velocities are zero. The approximation of a zero tangential velocity is adequate in all other cases tested, which can be explained by the correction nature of the unstructured MAC scheme.

When the tangential velocity is found from the normal velocity, the solution of an $s \times s$ matrix system, for every element, where s is the number of sides per element, is required. The techniques discussed in section (3.9) can be applied to solve this system. The over head they require is quite large considering such a small system is being solved and this is for every element every time step. The decision has therefore been taken to work out the inverse formula for each tangential velocity on all sides of each element. As the usual case is a 3×3 system, the formulas are very manageable. When merging occurs, there is the potential for s to become greater than three. For each possible number of sides per element, the inverse formulae would need to be worked out and a method of identifying which to use would be required. These formulae become very large with increasing s. Therefore, the decision has been made to limit the possible number of sides of an element to four. The inverse formulae for a 4×4 system are still manageable. The decision means that the merging routine cannot merge a quadrilateral element with another element. The merging routine does allow this, although it must be prevented through the use of a suitable merging factor.

5.2.3 Direct Solver Data Structures

The preferred choice for the solution of implicit systems is the use of a direct solver. As previously discussed the algorithms provided by the HSL group have been used to provide a well developed direct solver [86].

The choice of solver depends on the form of the implicit system matrix $\mathbf{A}$. The pressure correction equation produces a symmetric matrix and so the HSL MA57 [86] solver can be used. The MA57 solver requires the matrix being symmetric and so it is unsuitable for solving the system when implicit time stepping is used. When implicit time stepping is implemented an unsymmetric matrix occurs and so the MA41 solver is used instead.

Both MA57 and MA41 solvers require knowledge of the values in the matrix and of the right hand side of the implicit system. The matrix is sparse and should be stored as such. Three one dimensional arrays can be used to fully define a sparse matrix, one to store the non zero values of the sparse matrix, another to store the column locations of these values and the third to store the row locations. In the case of a symmetric matrix, only the upper triangular matrix needs to be stored, the lower off diagonal elements are duplicate values.

The direct solvers MA57 and MA41 are used as a black box, however, an outline of the general processes involved can be given [86]:

Initialise: The default values for components of the arrays that hold control parameters are set.

Analysis: The pattern of the matrix is analysed and the pivots for Gaussian Elimination are chosen. Only the pattern is required at this stage, not the matrix values.

Factorise: The matrix is factorised and so the actual values of the matrix are now required.

Solve: The factorisation is used to solve the system of equations. The right hand side of the linear system is required at this step.

Once the pattern of the non zero elements is known, the initialisation and analysis processes can take place. The factorise process can only take place after the matrix values have been assigned. In the case of the solution to the pressure correction equation and if a fixed time step is used with implicit time stepping, this is only required once. However, if adaptive time stepping is implemented in the implicit routine, the factorise step would need to occur every time iteration. The solve process is used whenever an update to the right hand side of the equation occurs thus every time iteration. The advantage of using the direct solvers is that the solve processes is the only stage that needs to occur every iteration.

To allow simple implementation of the direct solvers with the MAC solver, subroutines to perform the necessary operations are contained within a module. This module defines the data types for a spare matrix and each of the direct solvers.

5.3 Output

The main output of the code is the velocity and pressure variables and the vorticity at the end of the simulation, to be used in suitable visualisation software. All the software used by the author of this thesis requires is the output variables to be located at mesh nodes. The vorticity is already calculated at the nodes so does not pose a problem. However, the velocity and pressure must be interpolated back to the nodes. The velocity also needs extra consideration, the MAC solver only calculates the normal and tangential velocity components local to an edge. On output, they are required in a Cartesian coordinate system and so must be converted before the interpolation back to nodes can be made.

There are two forms of continual output calculated by the MAC solver. One to consider are the lift and drag coefficients calculated from the forces exerted on a submerged body. These coefficients evolve over time for an unsteady flow and serve as good validation mechanisms for various test problems. Also calculated at every iteration are the solution residuals. Residuals are required to provide a stop criteria for steady state flows. They are also a useful variable in unsteady flows, as they can be used to stop the solver if a solution has become unstable.

The theory of lift and drag forces and thus the formulation of the lift and drag coefficients will now be detailed. This is followed by the theory behind solution residuals.

5.3.1 Lift and Drag Forces

For flows around a submerged body a force is exerted onto the bodies surface due to pressure and viscous effects. This total force on a body can be calculated from the normal and tangential forces acting on each small section of the body. The normal forces are due to the fluid pressure and the tangential forces are due to the wall shear stress, τ_w. The wall shear stress takes into account the viscous effects on the body. The total force can be split into two components, the lift force, F_L and the drag force, F_D. Lift and drag forces act in different directions on the body, see figure (5.1). Lift acts in

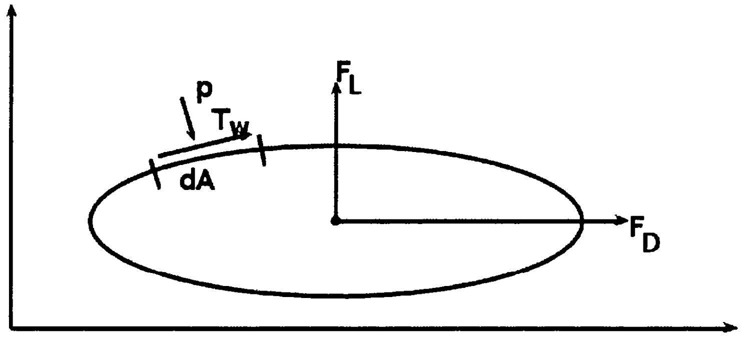


Figure 5.1: Lift and drag forces on a body in a fluid

the normal direction to the free stream velocity which can be considered as an up force moving the body up through the fluid. It its mainly driven by the pressure distribution rather than viscous effects. Drag acts in the direction of the free stream velocity, viscous effects will play an important part in the drag. Both lift and drag forces are calculated by resolving the pressure force and shear stress on the boundary of the submerged body.

If ω_A is the submerged body,$\partial\omega$ its surface, dA segment of the boundary, the lift and drag forces, F_L and F_D, can be written as,

$$F_L = \left(\int_{\partial\omega} -p(\boldsymbol{n}^w \cdot \boldsymbol{n}^\infty)dA + \int_{\partial\omega} \tau_w(\boldsymbol{t}^w \cdot \boldsymbol{n}^\infty)dA\right) \tag{5.1}$$

$$F_D = \left(\int_{\partial\omega} -p(\boldsymbol{n}^w \cdot \boldsymbol{t}^\infty)dA + \int_{\partial\omega} \tau_w(\boldsymbol{t}^w \cdot \boldsymbol{t}^\infty)dA\right) \tag{5.2}$$

Where the wall normal and tangent to the body are $\boldsymbol{n}^w$ and $\boldsymbol{t}^w$ respectively. The free stream normal is $\boldsymbol{n}^\infty$, with $\boldsymbol{t}^\infty$ being the free stream tangent, p is the pressure acting on the object, τ_w is the wall shear stress. See [101, 76, 75] for further information on lift and drag forces. In an incompressible fluid Perot [67] states that,

$$\tau_w = \mu\frac{\partial v}{\partial n},$$

is a suitable approximation for the wall shear stress. Where v is the velocity component tangential to the boundary. The derivative is therefore in the normal direction from the boundary. The full force F acting on the body is the sum of the lift and drag forces, $F = F_D + F_L$

Finding the lift and drag forces allows the evaluation of two new dimensionless variables, allowing ease of comparison with flow problems in the literature. To obtain the dimensionless values, known as the lift and drag coefficients, the lift and drag forces F_L and F_D are divided by,

$$\frac{1}{2}\rho u_0^2 A$$

where as before u_0 is a reference velocity and A is a reference area for the body. These give the lift and drag coefficients C_L and C_D as,

$$C_L = \frac{2L}{\rho u_0^2 A} \tag{5.3}$$

and

$$C_D = \frac{2D}{\rho u_0^2 A}. \tag{5.4}$$

The calculation of these values is implemented into the code by using each edge length as dA, the force integrals are then the sum of the pressure and viscous stress terms for each edge around the boundary. Using the boundary edges as each dA, allows the wall normals and tangents to be the predefined normal and tangents in the algorithm, although some checking of the direction is required. The pressure is the pressure value located at the boundary element centre neighbouring each side. The viscous stress term can be implemented by interpolating the velocity to the centre of the element and then converting its direction to the same as the current edge.

5.3.2 Residuals

A residual measures the difference in the solutions from one time step to the next. The initial step towards calculating it is to subtract the previous solution from the new one, producing a vector of differences. A residual gives an overall measure of this difference in solution. A vector of differences does not allow this and so it is necessary to calculate a norm for the vector. The norm assigns the vector with a positive value representative of its size, in the case of a residual, it gives an overall difference between the solutions.

There are many types of vector norms [78], to define these, let $\boldsymbol{v}$ be a real vector of length n. The first norm to consider is the 1-norm,

$$|\boldsymbol{v}|_1 = \sum_{i=1}^{n} |v_i|$$

which is the sum of the absolute value of all vector entries. Next is the 2-norm also known as the L_2 norm or the Euclidean norm,

$$|\boldsymbol{v}|_2 = \left(\sum_{i=1}^{n} |v_i|^2 \right)^{\frac{1}{2}}.$$

A more general version of the 2-norm is the p-norm, where the 2 in the above is replaced by p, which can be any positive integer. This is defined as,

$$|\boldsymbol{v}|_2 = \left(\sum_{i=1}^{n} |v_i|^p \right)^{\frac{1}{p}}.$$

The last norm to define is the ∞-norm,

$$|\boldsymbol{v}|_2 = \max_{i=1}^{n} |v_i|$$

which is simply the maximum value in the vector.

The chosen norm to calculate residuals is the 2-norm, giving an indicator of the amount the solution is changing each iteration. For a converging solution, this amount should decrease each iteration, although there is no guarantee that the solution to which the solver is converging to is the correct solution. In the cases above the vector $\boldsymbol{v}$ is the difference between the new solution and the old solution. The residuals can be calculated for any three of the solution variables, in the unstructured MAC case this is chosen to be the normal velocity component. The velocity has higher order of accuracy than the pressure and so lower residuals are reached. Since the tangential velocity is recovered from the normal velocity, their residuals are very similar.

To give a clearer indicator of the amount a solution is converging, the base ten logarithmic difference is taken with the initial residual. This compares the residual with the initial, in a manner that shows how many orders of magnitude a solution has converged.

Chapter 6

Benchmark Results

In order to validate the unstructured MAC scheme, a set of standard CFD benchmark problems are chosen. Initially the lid driven cavity problem was simulated. The lid driven cavity is a steady state problem which is historically used to validate CFD results, where there are extensive results given in the literature for many different Reynolds numbers [1]. A short study on using a skewed mesh with the lid driven cavity problem has also been carried out. This test case demonstrated the affects of non-orthogonality in the meshes. The second benchmark problem is the flow over a backward facing step, again a steady state problem. This requires the need for suitable inflow and outflow boundary conditions. The third benchmark is flow around a circular cylinder, where there are several comparisons that can be made using this example. For the case of inviscid flow around a circular cylinder, there exists an analytical solution for the pressure coefficient on the cylinder surface. Unsteady flows can also be validated using this example, where unsteady effects occur when the Reynolds number is greater than 50. Results are compared to literature for several Reynolds numbers that demonstrate unsteady flow.

The validation of the algorithm against past results and experiments is just the initial stage. The purpose of the proposed algorithm is to produce results with a shorter CPU time than an alternative CFD code used at Swansea University. Therefore, a comparison is made with the in-house incompressible finite volume code, not only are the CPU times compared but also the results show an interesting outcome.

This chapter will present results for the various test cases and compare them with results from the literature. The order of these results is as follows: The lid driven cavity; flow over a backward facing step; inviscid flow around a circular cylinder; and unsteady viscous flow around a circular cylinder.

6.1 The Lid Driven Cavity

The lid driven cavity has long been a standard test problem for steady state flows. The domain consists of a 1×1 cavity, with a moving lid of unit velocity. To specify this problem in a numerical framework, the moving lid is achieved by setting the tangential velocity on the top boundary equal to one. The wall boundary condition is set to all other boundaries and so both velocities are zero. Figure (6.1) displays the domain set up, note that here u and v are the velocities in Cartesian coordinates hence u being the tangential component to the top edge. Presented are results for flows with three different Reynolds

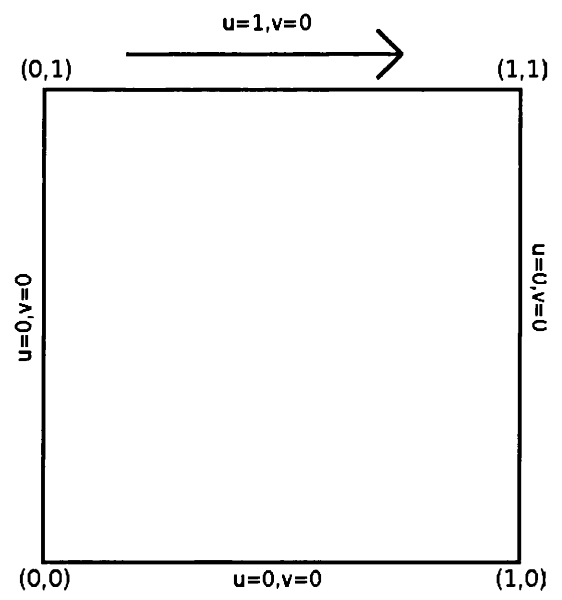


Figure 6.1: Domain for the lid driven cavity flow problem

numbers, 100, 400 and 1000 and compared to the benchmark solution of Ghia et.al. [1]. The standard procedure to compare results in the lid driven cavity is to compare velocity profiles up and across the centre of the domain. The horizontal velocity profile is plotted up the centre of the domain, i.e $x = 0.5$ and y coordinate varies. The vertical velocity profile is plotted across the centre of the domain, i.e $y = 0.5$ and coordinate x varies.

Described in previous sections are various time stepping methods and implicit solver routines that could be used in the implementation of the unstructured MAC algorithm. To determine the best possible combination, the MAC algorithm is compiled into four configurations. The four configurations that will be used to demonstrate the algorithm are:

1. An explicit time stepping routine with a conjugate gradient solver of the pressure

correction equation.

2. An explicit time stepping routine with a direct solver for the pressure correction equation.

3. A implicit Euler time stepping routine solver via a direct method with a direct solver for the pressure correction equation. Using the semi-implicit form of the discretisations.

4. The crank Nicholson time stepping routine solved via a direct method with a direct solver for the pressure correction equation. Using the semi-implicit form of the discretisations.

In the explicit time stepping routines the time step is calculated each iteration using equation 3.26. In the case of semi-implicit time stepping to increase computational efficiency, the time step is provided at the start of the simulation and is fixed through all iterations.

The lid driven cavity is a simple steady state problem to simulate, there are many well documented results to compare to, the standard of which are those by Ghia. et. al [1]. For this reason, the simulations are run with the four afore mentioned MAC configurations, the results from which will help determine which configurations are potentially better.

Initially an unstructured triangular mesh created using the stitching algorithm is used, it can be seen that an ideal Delaunay mesh in the centre of the domain is linked to a mesh that is fitted to the boundary, see figure (6.2). The mesh consists of 3670 elements with 40 nodes along each boundary. The Voronoï dual mesh created from connecting the triangle circumcentres is shown in figure (6.3).

Results will now be presented for the three Reynolds numbers using the four configurations. run-times and time step sizes will be compared as well as each final solution to the benchmark. All test cases are run on an AMD opteron processor. For each case the residuals are allowed to converge five orders of magnitude.

The simulation with $Re = 100$ is the simplest to obtain results for, there are less re-circulation areas produced due to less viscous effects in the boundary region, making accurate solutions obtainable on very coarse meshes. Run-times for each configuration are shown in table (6.1). It is also of interest to calculate the run-time per iteration, this gives an indicator of where the extra computation time in each method is required. In this case, using the CG method greatly increases the the run-time per iteration, since

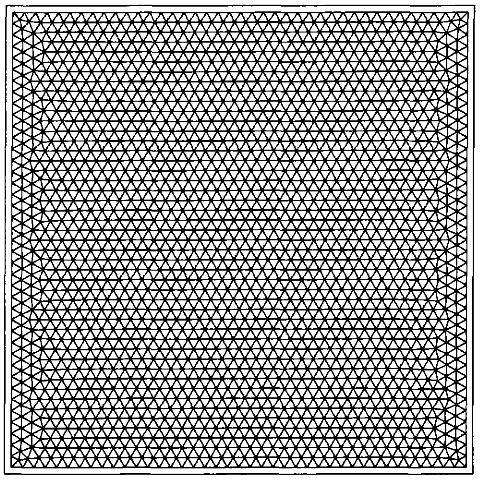


Figure 6.2: Delaunay mesh for lid driven cavity domain

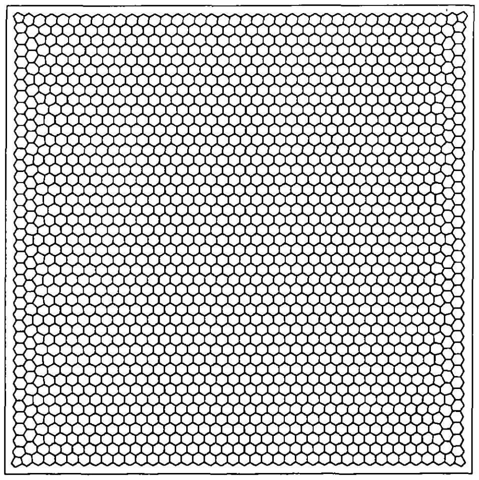


Figure 6.3: Voronoï mesh for lid driven cavity domain

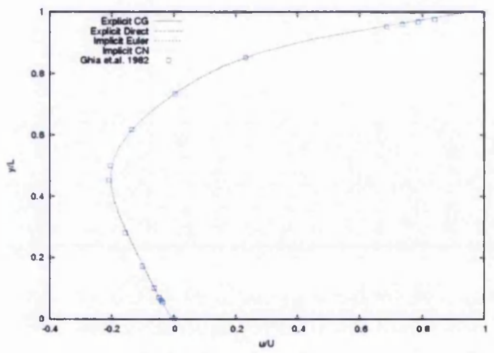
	Run Time	Time Step	Iterations	Time per iteration
Explicit CG	13.983s	0.05	834	0.0168s
Explicit Direct	3.96s	0.05	834	0.00475s
Implicit Euler	0.77s	0.5	60	0.0128s
Implict CN	1.238s	0.1	128	0.00967s

Table 6.1: run-time and time step size comparison for the lid driven cavity problem with a Reynolds number of 100

the time stepping routine is explicit it still requires a similar number of iterations to the explicit direct case. All results compare well to those found in the literature, see figure (6.4). The MA57 direct solver outperforms the CG method for solution of the pressure correction equation. Which is too be expected as this is a highly developed code. The implicit scheme under performs the explicit scheme for this test case, the results from other test cases will help to determine a reason. To demonstrate the re-circulation region, the results for each velocity component for the full domain can be viewed in figures (6.5) and (6.6).

As results for each case are very similar, only those from the explicit direct case are presented. The next simulation tests the MAC solvers capabilities at a higher Reynolds number, thus, $Re = 400$ is simulated. The various run-times are given in table (6.2), again demonstrating that the use of the CG method with the explicit scheme and semi-implicit schemes are less efficient for this problem.

For this case the viscous regions on the boundaries are larger and so it is reasonable to expect that a finer mesh in the boundary regions will be required. However, the


(a) Horizontal velocity

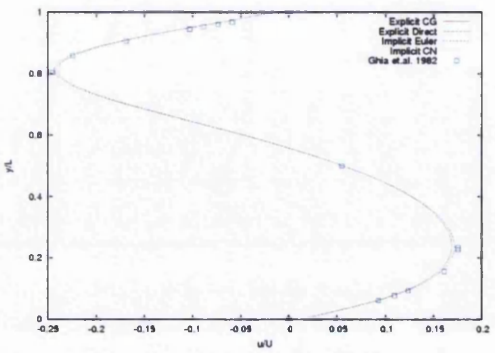

(b) Vertical velocity

Figure 6.4: Lid driven cavity with $Re = 100$, results comparison to literature [1].

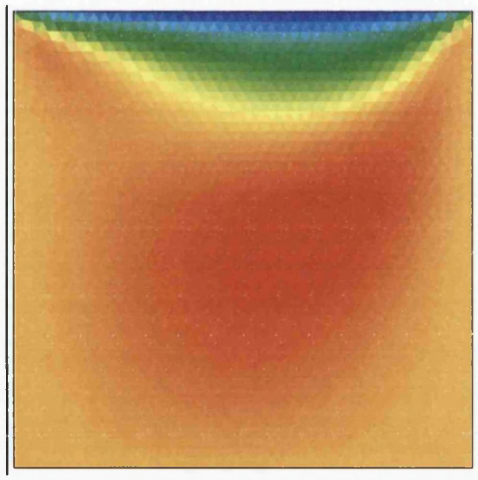


Figure 6.5: Horizontal velocity for $Re = 100$. Colour scheme is red=high, blue=low.

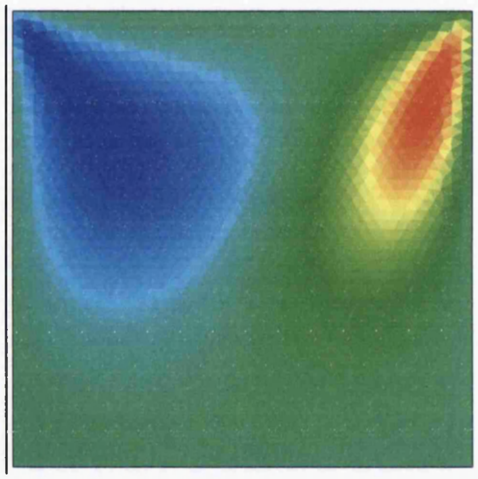


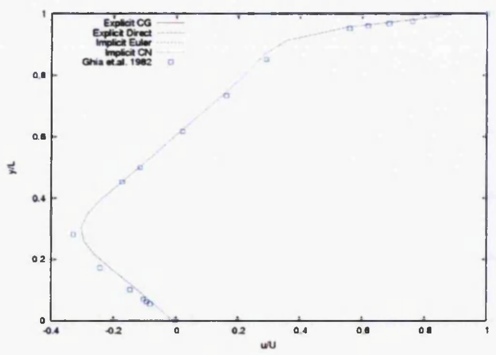
Figure 6.6: Vertical velocity for $Re = 100$. Colour scheme is red=high, blue=low.

	Run Time	Time Step	Iterations	Time per iteration
Explicit CG	26.072s	0.023	1087	0.0240
Explicit Direct	4.905s	0.023	1911	0.00257
Implicit Euler	5.795s	0.045	621	0.00933
Implict CN	5.667s	0.04	685	0.00873

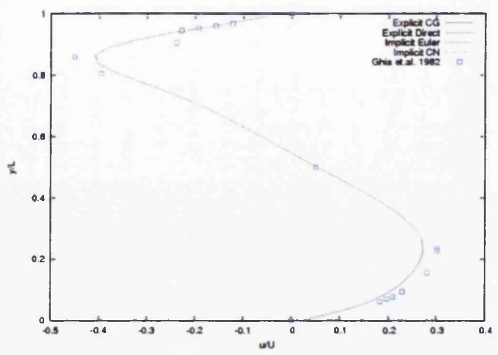
Table 6.2: run-time and time step size comparison for the lid driven cavity problem with a Reynolds number of 400

case is still run using the mesh given in figure (6.2). Figure (6.7) shows the velocity profiles compared to that of Ghia, the results still compare well even on this relatively coarse mesh, where only a slight discrepancy occurs. It is expected that increasing the

Reynold number further will produce results that fail to be accurate enough. All the



(a) Horizontal velocity



(b) Vertical velocity

Figure 6.7: Lid driven cavity with $Re = 400$, results comparison to literature [1]

configurations produced a similar result. For a view on the flow for the entire domain and the different re-circulation pattern produced at this Reynolds number, figures (6.8) and (6.9) show the horizontal and vertical velocity components.

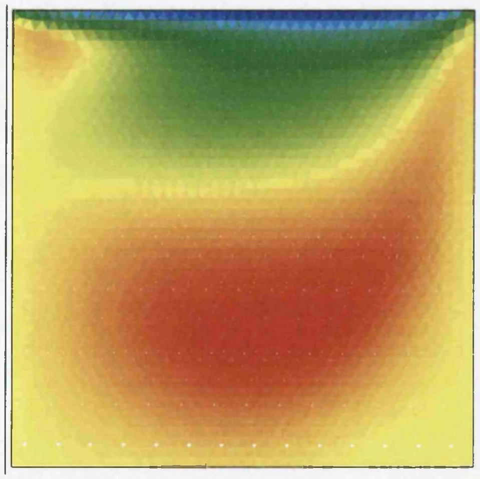


Figure 6.8: Horizontal velocity for $Re = 400$. Colour scheme is red=high, blue=low.

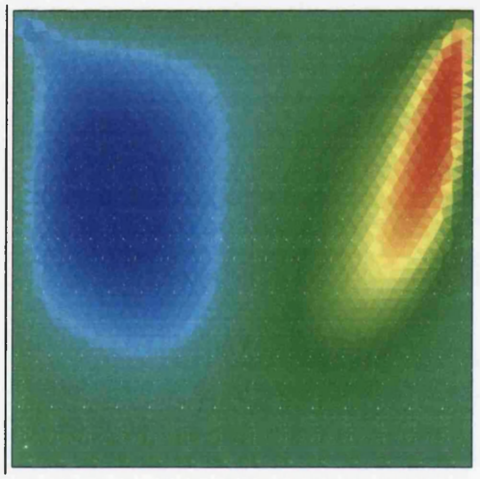
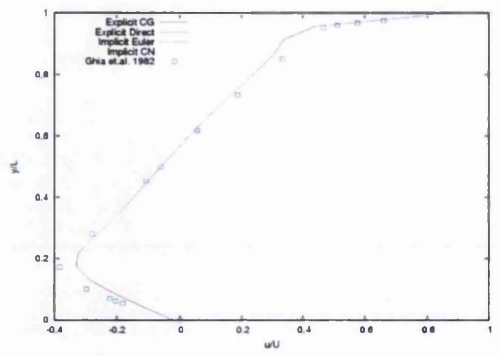
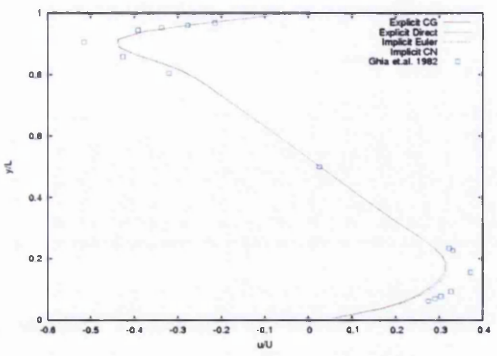


Figure 6.9: Vertical velocity for $Re = 400$. Colour scheme is red=high, blue=low.

To further test the scheme and the conjecture that for higher Reynolds number the mesh is not fine enough, a simulation is run with $Re = 1000$. The results fail to compare well to the Ghia solution in the extreme regions of the flow, flow speeds do not achieve the higher and lower limits, as observed in figure (6.10). To further asses the efficiency of each algorithm configuration, a comparison of the run-times is still



(a) Horizontal velocity



(b) Vertical velocity

Figure 6.10: Lid driven cavity with $Re = 1000$, results comparison to literature [1]

	Run Time	Time Step	Iterations	Time per iteration
Explicit CG	1m 7.325s	0.0096	2939	0.0229
Explicit Direct	13.653s	0.0096	2994	0.00456
Implicit Euler	15.116	0.021	1608	0.0094
Implict CN	14.757	0.023	1493	0.0099

Table 6.3: run-time and time step size comparison for the lid driven cavity problem with a Reynolds number of 1000

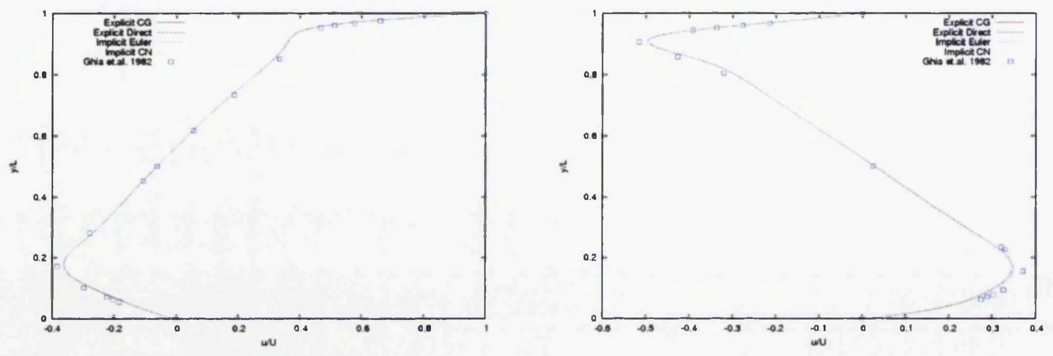
made, see table (6.3). To confirm the cause of the lack of accuracy as being the use of a coarse mesh, the simulation is re-run using a finer mesh, consisting of right angled triangle elements. Although this is a structured mesh which does not require the devised unstructured algorithm, it has a finer resolution than the previous mesh and can still be used to demonstrate why accuracy was not achieved. The use of the right angled triangles also allowed a test for the codes merging capabilities. The mesh consists of 160 boundary nodes per side resulting in 51200 triangular elements, which the code successfully merges to half the amount of quadrilateral elements. It is still of interest to compare the run-times for this case, the number of elements in the mesh is much greater having an amplifying effect on the differences in run-time, see table (6.4). From

	Run Time	Time Step	Iterations	Time per iteration
Explicit CG	32m 31.212s	0.00674	3719	0.525s
Explicit Direct	3m 38.414s	0.00620	4122	0.0530s
Implicit Euler	4m 0.241s	0.01	2797	0.0859s
Implict CN	4m 19.316s	0.009	3146	0.0824s

Table 6.4: run-time and time step size comparison for the lid driven cavity problem with a Reynolds number of 1000

these results it is clear that the conjugate gradient as coded within the MAC method is significantly slower when compared to the MA57 direct solver. A run-time of nearly 30 minutes longer is recorded for the $Re = 1000$ case.

The extra cost in time required for the larger mesh is proved necessary when the results are compared to the literature. The velocity profiles provide a near perfect match, figure (6.11). For completeness, the velocity components for the entire domain on the



(a) Horizontal velocity (b) Vertical velocity

Figure 6.11: Lid driven cavity with $Re = 1000$, results comparison to literature

finer mesh are again shown, see figures (6.12,6.13). Once again a more complex re-

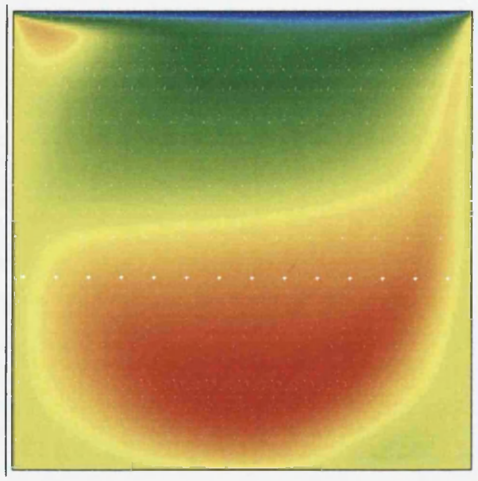


Figure 6.12: Horizontal velocity for $Re = 1000$. Colour scheme is red=high, blue=low.

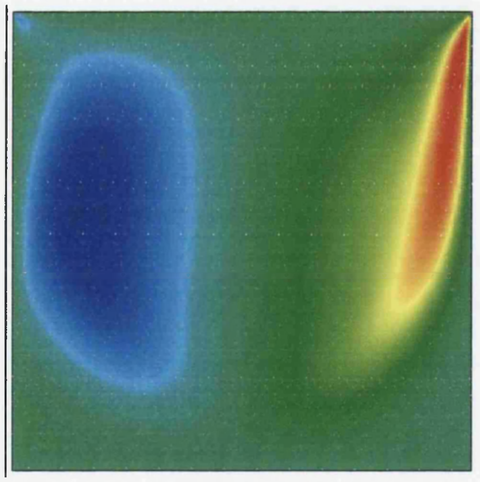


Figure 6.13: Vertical velocity for $Re = 1000$. Colour scheme is red=high, blue=low.

circulation region can be seen.

From analysing the run-time results for all three Reynolds numbers some conclusions can be drawn. The definition of the time step size states that as the Reynolds number increases, the size of the viscous time step also increases. So for higher Reynolds numbers the time step size is governed by the convective term. This is observed in the results, where the higher Reynolds number test cases showed that the time step size is similar for both the explicit and implicit schemes. The time steps given for the explicit schemes are the final one observed in the simulations. In this case the time stepping is adaptive and has the potential to change each iteration, however, once the solution tends to convergence, variations in the time step have usually ceased. The implicit time step is specified at the start of the solution, based on the explicit time step formula but allowing higher CFL numbers. Both time stepping routines required testing of suitable CFL numbers to ensure the time step being used was the smallest possible to produce a stable solution. The quoted time steps include the scaling with the CFL number. They therefore represents the maximum time step sizes that were used.

To fully capture the effects of the moving wall, there needs to be some explicit treatment of the viscous effects on the boundary. Thus, an explicit viscous calculation is still contained in the solver formulation, which not only has influence on the time step size but also requires extra computational effect to compute the values.

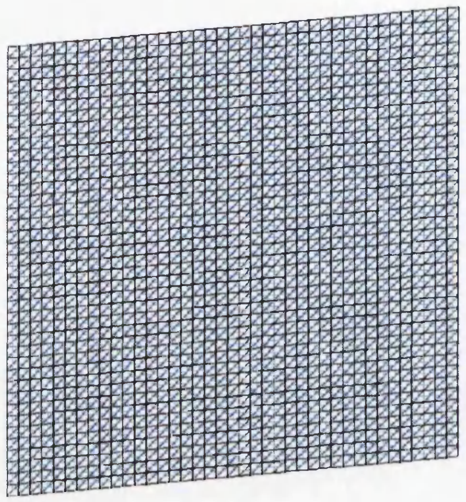
In general, the semi-implicit case requires extra computational effort to invert the matrix system and so much fewer time steps are required to reach convergence. The restrictions on the time step size required for a stable solution mean that even though the number of iterations may be halved, it is still not enough to outweigh the computational effort required to solve the system.

The present results suggest that in test cases similar to the lid driven cavity, i.e problems that contain a moving wall, a small domain mesh and are steady state, may produce solutions faster using the explicit time stepping scheme.

6.2 Lid Driven Cavity Skewed Mesh Study

To access the effects of using an non orthogonal mesh, a skewed cavity is used. The skewed cavity changes the angle of the horizontal edges. Two cases are tested, one changing the angle to 5^o and the other changing the angle to 20^o. Changing the angle has two affects on the mesh quality; it removes orthogonality by changing the angle between intersecting Voronoï and Delaunay edges; and creates bad elements so that Delaunay triangles circumcentres lie outside the triangles.

The first case to look at is skewing the mesh by an angle of 5^o. See figure 6.14 for the mesh when merging is not used. Figure 6.14(b) shows that the circumcentres are outside


(a) Skewed mesh by an angle of 5^o for lid driven cavity

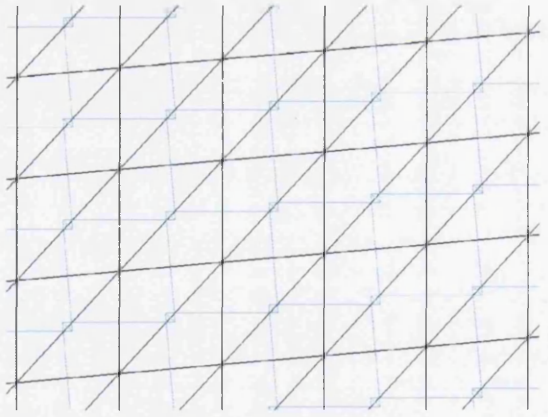

(b) Close up of skewed mesh,skewed by an angles of 5^o, for lid driven cavity

Figure 6.14: Skewed lid driven cavity mesh by angle of 5^o when merging is not used

of the triangles. Using the test case of a lid velocity of one and a Reynolds number of 100, if these centres are used in the unstructured MAC algorithm, a solution to the lid driven cavity problem is unstable. Figures 6.15, 6.16 and 6.17 show the solution for the pressure, horizontal velocity and vertical velocity respectively just before the solution values become exponential. For the mesh in figure 6.14(a) the Voronoï edges are short enough that mesh merging takes place giving the mesh shown in figure 6.18. In

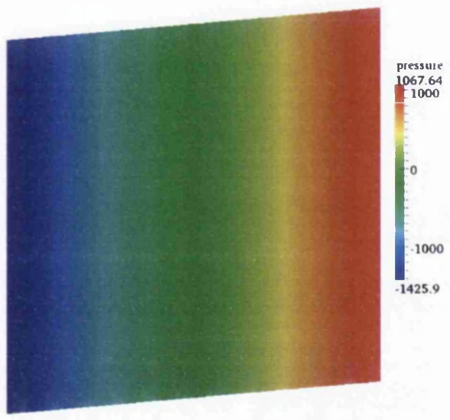


Figure 6.15: Pressure for the skewed lid cavity case using a mesh that is skewed by 5^o and mesh merging is not used

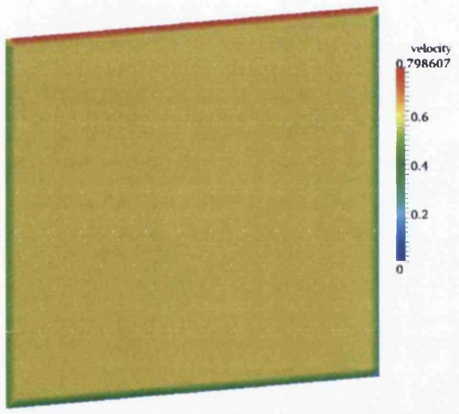


Figure 6.16: Horizontal velocity for the skewed lid cavity case using a mesh that is skewed by 5^o and mesh merging is not used

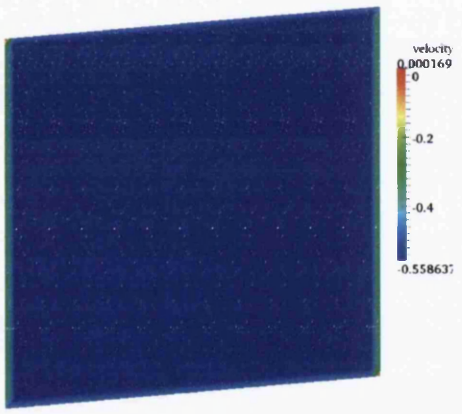
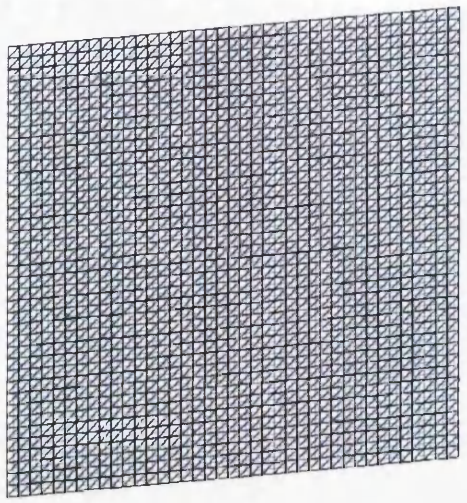


Figure 6.17: Vertical velocity for the skewed lid cavity case using a mesh that is skewed by 5^o and mesh merging is not used

figure 6.18 the Delaunay elements are shown as the un-merged to attempt to show the position of the new cell centre in relation to the old. The elements are merged within the code and the diagonal edges removed. This means that the element centre, the Voronoï vertex, lies within the merged cell. If mesh merging is used then a stable solution can be found. Figures 6.19, 6.20 and 6.21 show the solution of the pressure, horizontal velocity and vertical velocity respectively after it has converged five orders of magnitude. This case demonstrates the importance of the location of the Voronoï vertex. After merging, the vertex is now inside the Delaunay cell and the and a stable solution can be found.


(a) Skewed mesh by an angle of 5^o for lid driven cavity

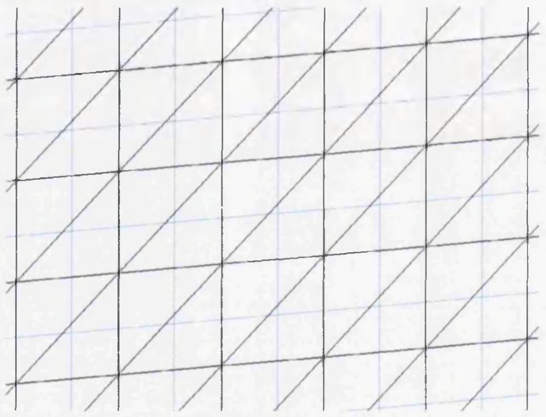

(b) Close up of skewed mesh by an angle of 5^o for lid driven cavity. The Delaunay cells are shown are not merged but are merged in the code. The Voronoï cells in blue are merged to quadrilaterals.

Figure 6.18: Skewed lid driven cavity mesh by angle of 5^o when merging is used

The solution shown in figures 6.19, 6.20 and 6.21 visually looks close to those using the non skewed mesh at Re=100 even when the intersection between the Voronoï and Delaunay edges are not at right angles.

A section skewed mesh is also tested. In this case the mesh is skewed by an angle of 20^o, see figure 6.22(a) and figure 6.22(b). Figure 6.22 shows meshes with non merged cells. In this case the Voronoï edges are not short enough so that the elements can be merged. A stable solution is not possible on this mesh if the circumcentres are used

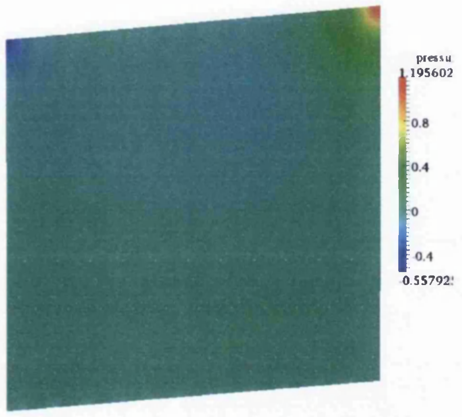
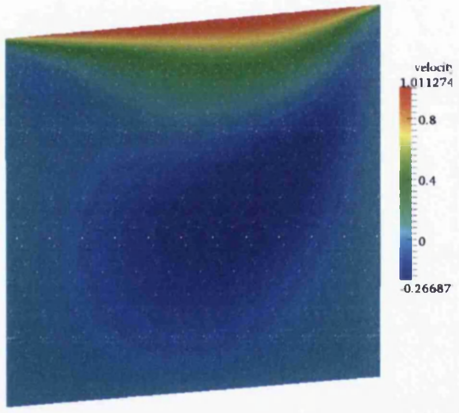


Figure 6.19: Pressure for the skewed lid cavity case using a mesh that is skewed by 5^o and mesh merging is used



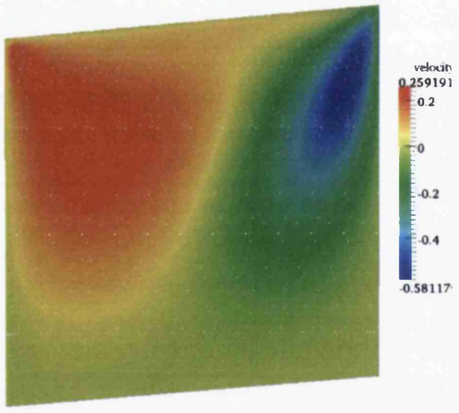
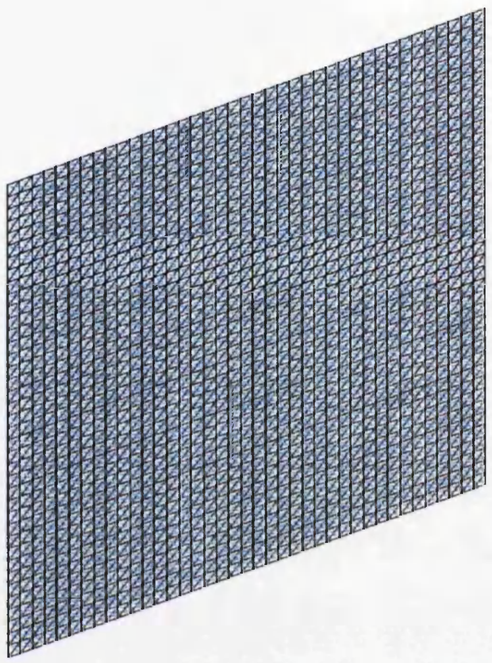


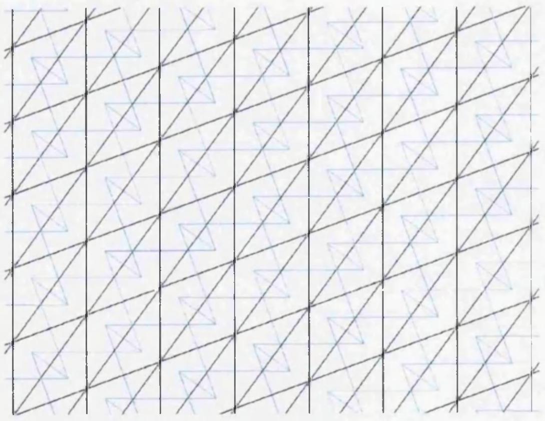
Figure 6.20: Horizontal velocity for the skewed lid cavity case using a mesh that is skewed by 5^o. The mesh is merged throughout so that two triangles that are close to being right angled merged to quadrilaterals.

Figure 6.21: Vertical velocity for the skewed lid cavity case using a mesh that is skewed by 5^o. The mesh is merged throughout so that two triangles that are close to being right angled merged to quadrilaterals.

as the location of the Voronoï vertex. Figures 6.23, 6.24 and 6.25 show the unstable solution before the values become exponential. If the Voronoï vertex is moved from the circumcentre to the average between the barycentre and circumcentre then a stable solution is possible. Figures 6.26, 6.27 and 6.28 shows this stable solution. From the figures it is apparent that the solution accuracy is not as good as that with the skewed mesh of 5^o, this is possibly an artifact of the skewed mesh.



(a) Skewed mesh by an angle of 20^o for lid driven cavity



(b) Close up of skewed mesh by an angle of 20^o for lid driven cavity

Figure 6.22: Skewed lid driven cavity mesh by angle of 20^o when merging is not used

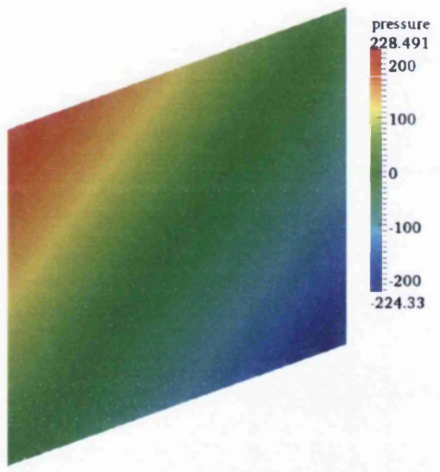


Figure 6.23: Pressure for the skewed lid cavity case using a mesh that is skewed by 20^o and mesh merging is not used

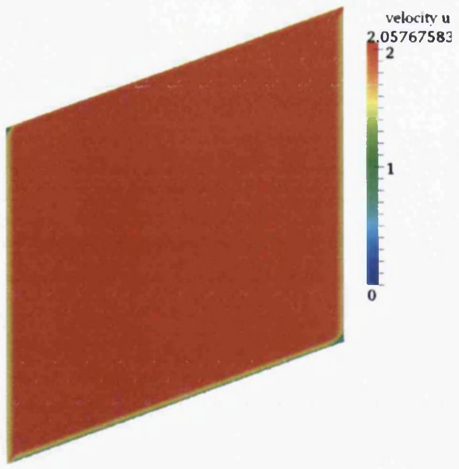


Figure 6.24: Horizontal velocity for the skewed lid cavity case using a mesh that is skewed by 20^o. The triangular elements in the mesh are not merged to quadrilaterals.

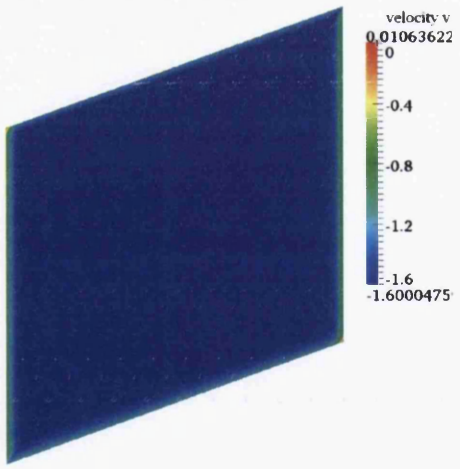


Figure 6.25: Vertical velocity for the skewed lid cavity case using a mesh that is skewed by 20^o. The triangular elements in the mesh are not merged to quadrilaterals.

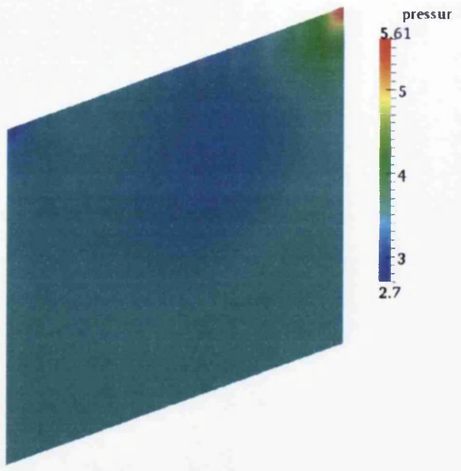
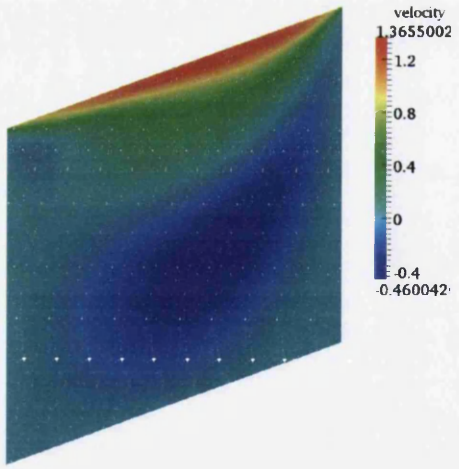


Figure 6.26: Pressure for the skewed lid cavity case using a mesh that is skewed by 20^o and the Voronï vertex is moved to the average between the Delaunay barycentre and the circumcentre.



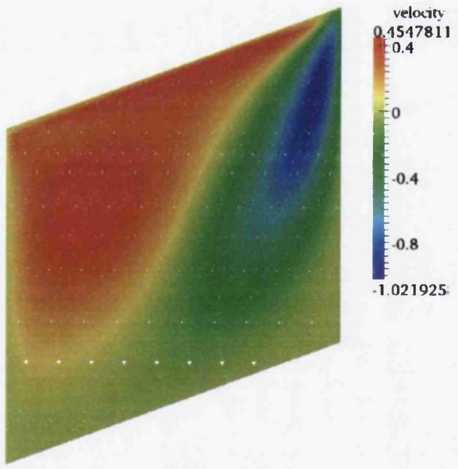


Figure 6.27: Horizontal velocity for the skewed lid cavity case using a mesh that is skewed by 20^o and the Voronï vertex is moved to the average between the Delaunay barycentre and the circumcentre.

Figure 6.28: Vertical velocity for the skewed lid cavity case using a mesh that is skewed by 20^o and the Voronï vertex is moved to the average between the Delaunay barycentre and the circumcentre.

6.3 Flow Over a Backward Facing Step

The second test case is the flow over a backward facing step. This test problem is again steady state but is more difficult to simulate than the lid driven cavity problem. The increase in difficulty is due to the presence of an inflow and outflow boundary condition, the first test problem to include such boundaries. The geometry, shown in figure (6.29), uses a step height of h, and an inflow channel of height h and length $6h$. The outflow

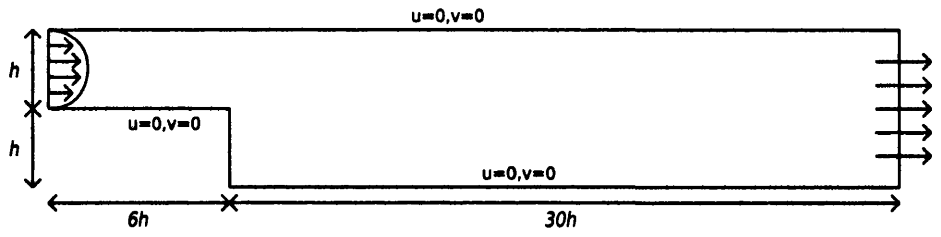


Figure 6.29: Domain for flow over a backward facing step

is located a distance of $30h$ from the position of the step. This is a fairly short distance from the step, thus creating a good test for any outflow boundary condition formulation.

For this benchmark problem, the standard case to compare to in the literature is that of Armaly et.al [102]. Reported in the literature are both experimental and numerical results, the experimental results providing a desirable comparison for the unstructured MAC algorithm. The inflow velocity is required to be parabolic with a maximum velocity comparable to the results given in the literature, the value of which is different for each Reynolds number [102].

There are two Reynolds numbers compared for the backward step case, the first being $Re = 100$ and the second $Re = 389$. No results are given for higher Reynolds numbers as when higher Reynolds numbers where used the solution became unstable. This could be due to the the flow to becoming three dimensional [102].

To determine the inflow parabola, a maximum inflow velocity is obtained from the literature, this is converted to fit the dimensionless scales in the back facing step mesh used in the simulation. The mesh used for all simulations of this test example, contains a refined region around the step, see figure (6.30). A re-circulation region should occur in the back of the step region, therefore, the refinement of the mesh is needed in this area to ensure accurate capturing of all fluid flow features.

The numerical results are then compared to the literature. The scale used in the unstructured MAC code is $h = 1$, whereas, in the literature $h = 0.011$. The distance from the step is scaled by the height of the step h and the horizontal velocity is given

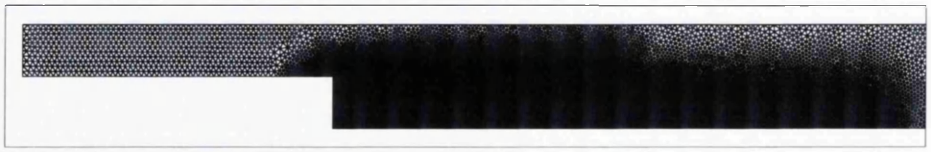


Figure 6.30: Section of mesh used to simulate flow over a backward facing step. Displays the refined region around the step. The mesh contains 53,427 elements, 27,287 nodes and 1,145 boundary nodes.

in metres per second. In the results the height remains in dimensionless form allowing comparison in the two scales that have been used.

The test case of $Re = 100$ is simulated using the explicit time stepping scheme, where results are displayed in figure (6.31), with the literature results given by the green points and the numerical results by the red line. Results compare well to the literature,

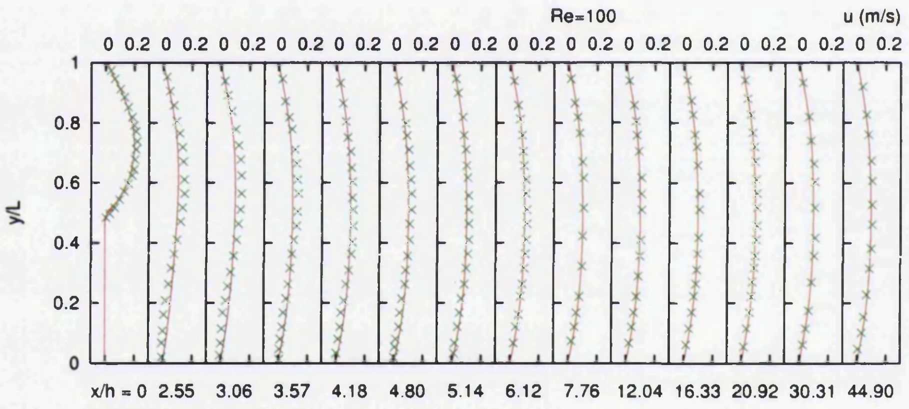


Figure 6.31: Comparison to literature for flow over a backward facing step with $Re = 100$. x/h=0 is the at the location of the step.

considering that the comparison is based on digitised graph data from the literature. The results for all variables are given in figures (6.32,6.33,6.34), to give a view of the variables distribution for the entire domain. A Reynolds number of 100 is the easier

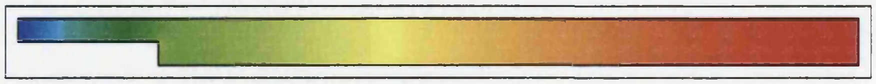


Figure 6.32: Pressure for flow over a backward facing step with $Re = 100$. Colour scheme is red=high, blue=low.

example, where there are less viscous effects in the solution so only the explicit time stepping routine has been used to simulate the results.

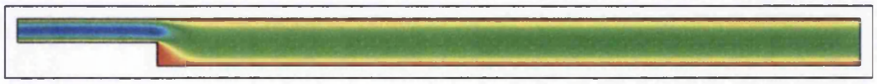


Figure 6.33: Horizontal velocity for flow over a backward facing step with $Re = 100$. Colour scheme is red=high, blue=low.

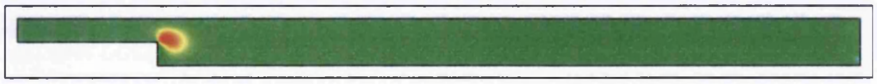


Figure 6.34: Vertical velocity for flow over a backward facing step with $Re = 100$. Colour scheme is red=high, blue=low.

To further demonstrate the unstructured MAC codes capabilities, a higher Reynolds number of $Re = 389$ is used. This case has more viscous effects and the re-circulation region is larger, providing a good test case for the outflow boundary condition, as fluid features are closer to the boundary. As this is a more complex case, it is also used to test the various time stepping configurations. Horizontal velocity profiles are plotted against the results reported in the literature in the same format as $Re = 100$, results compare well. All the required more complex flow characteristics are displayed, see figure (6.35). Figures (6.36,6.37,6.38), give a view of the variables distribution for the entire domain.

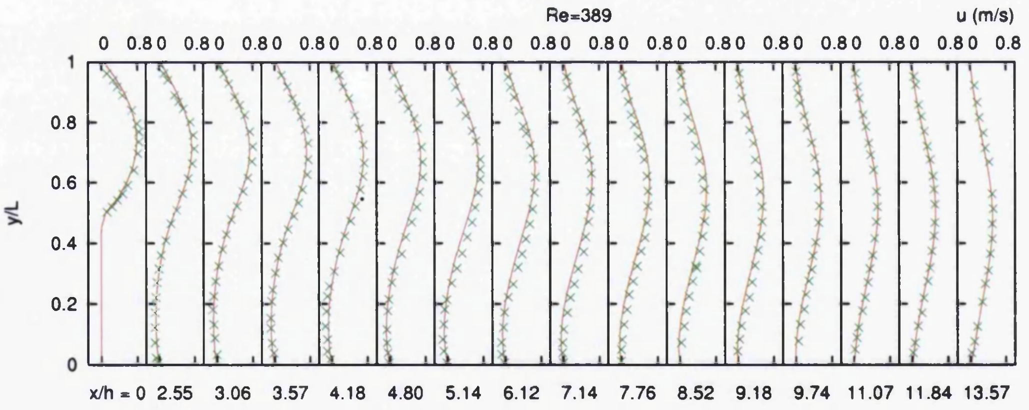


Figure 6.35: Comparison to literature for flow over a backward facing step with $Re = 389$. x/h=0 indicates the location of the step.

The run-times for the four configurations for the unstructured MAC scheme can also be compared. The mesh used for both cases is the same and the Reynolds number does not differ by a huge amount and so the comparison is just given for the case where $Re = 389$, see table (6.5). For the steady state case the explicit scheme outperforms the implicit case due to the time step size increase not being enough to counteract the

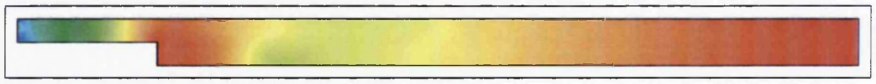


Figure 6.36: Pressure for flow over a backward facing step with $Re = 389$. Colour scheme is red=high, blue=low.

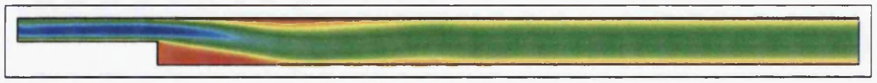


Figure 6.37: Horizontal velocity for flow over a backward facing step with $Re = 389$. Colour scheme is red=high, blue=low.

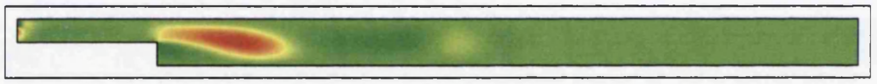


Figure 6.38: Vertical velocity for flow over a backward facing step with $Re = 389$. Colour scheme is red=high, blue=low.

	Run Time	Time Step
Explicit CG	1h 4m 48s	0.0362
Explicit Direct	268.341s	0.0362
Implicit Euler	328.709	0.07
Implict CN	288.799	0.08

Table 6.5: Run-time and time step size comparison for flow over a backward facing step at a Reynolds number of 389

increase in the time required per iteration. It is interesting to see that the conjugate gradient method to solve the Poisson equation is exceedingly slow, with a run time of over an hour compared to ones of around five minutes. Therefore, all future comparison will not include the conjugate gradient method.

6.4 Inviscid Flow Around a Circular Cylinder

Inviscid flow around a circular cylinder is a steady state problem that has an analytic solution for the pressure exerted on the cylinders surface [101]. For an inviscid flow simulation, the viscous term is dropped from the discrete equations. Comparing the inviscid simulation result to the analytic will not confirm accuracy of the full viscous equations but will confirm that all other processes in the algorithm are being evaluated correctly.

Before the numerical results are presented, the analytical solution for the surface pressure p_s is detailed. This can be expressed as [101],

$$p_s = p_0 + \frac{1}{2}\rho u_0^2(1 - 4sin^2\alpha), \tag{6.1}$$

where p_0 is the initial or reference pressure value for the domain, u_0 a reference velocity and α the angle from zero around the cylinder as shown in figure (6.39). The value for

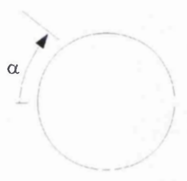


Figure 6.39: Angles α starting from facing edge of cylinder

the surface pressure can be rearranged to give the pressure coefficient in dimensionless form as,

$$\frac{p_s - p_0}{\frac{1}{2}\rho u_o^2} = (1 - 4sin^2\alpha). \tag{6.2}$$

The presence of an analytic solution, not only confirms the accuracy of the inviscid part of the MAC scheme but it can be used to determine the solution accuracy dependence on the mesh resolution. This allows a mesh convergence study to take place. A mesh convergence study should demonstrate that a solution gets closer to the analytical solution with finer mesh resolutions.

To carry out this study, suitable meshes are required. These meshes were constructed by'growing' equilateral triangles out from the cylinder of diameter one. Three meshes with a varying the number of nodes on the cylinder were produced, the first with 100 nodes, the second with 200 and the third with 400 nodes. Figure (6.40) displays an example of one of these meshes. The largest of the three meshes, 400 nodes around the cylinder has 162,400 elements and 81600 nodes, requiring approximately two minutes to converge the pressure seven orders of magnitude. As expected figure (6.41) shows that the accuracy increases as the mesh resolution increases. The CPU times presented are when using an explicit time stepping procedure. No implicit treatment of the convective term has been implemented, therefore no implicit results are possible for this case.

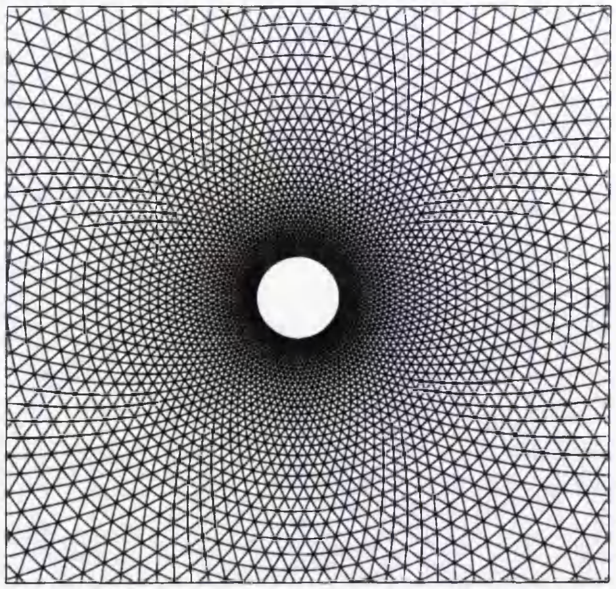


Figure 6.40: Mesh for inviscid flow example with 100 nodes around the cylinder

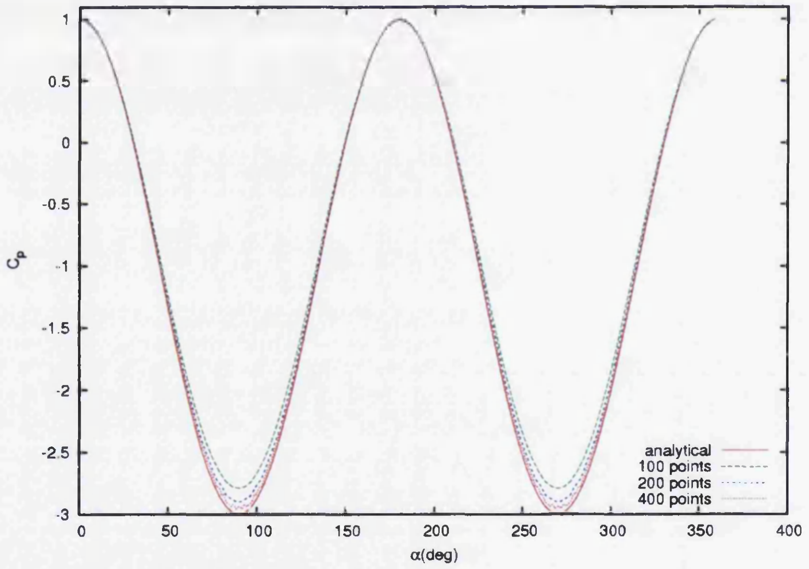
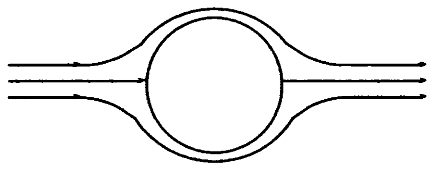


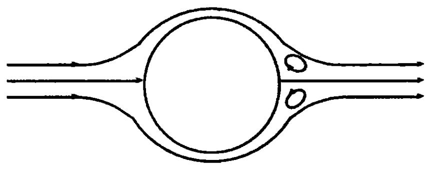
Figure 6.41: Mesh for inviscid flow example with 100 nodes around the cylinder

6.5 Unsteady Viscous Flow Around a Circular Cylinder

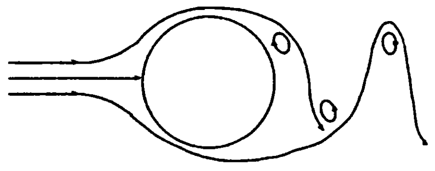
For viscous flow around a circular cylinder, steady and unsteady state solutions are possible. The type of flow that will be exhibited will depend on the Reynolds number. Figure (6.42) shows the flow characteristics for the various Reynolds numbers [103]. The previous examples all demonstrate that the unstructured MAC algorithm is capable



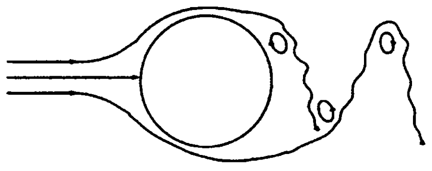
(a) $Re < 5$, un-seperated flow



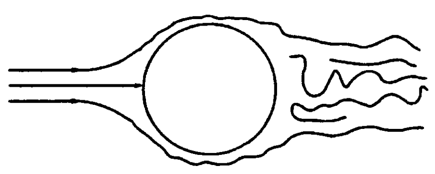
(b) $5 \leq Re < 40$, fixed pair of vortices in the wake.



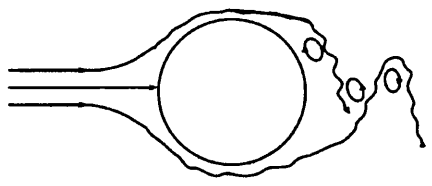
(c) $40 \leq Re < 150$, laminar vortex shedding.



(d) $150 \leq Re < 300$, transition range to turbulence in vortex. $300 \leq Re < 3 \times 10^5$, vortex street is fully turbulent.



(e) $3 \times 10^5 < Re < 3.5 \times 10^6$, fully turbulent flow with no apparent vortex street.



(f) $3.5 \times 10^6 \leq Re < \infty$, re-establishment of a turbulent vortex street. Has a turbulent boundary layer and thinner wake.

Figure 6.42: Flow characteristics for various Reynolds numbers for flow around a circular cylinder.

of modelling steady state flows and so it is unsteady flows that are now of interest. At present there is no turbulence model incorporated into the solver and so it is the range of Reynolds numbers that demonstrate a laminar vortex street that are considered. It will also be shown that successful solutions can be obtained in the transition to turbulence range. Therefore, for the viscous case, flow at two Reynolds numbers will be modelled, $Re = 100$ and $Re = 200$. For a Reynolds number of 100, a structured mesh like those used for the inviscid case will be used. To later demonstrate the MAC solver capabilities, a fully unstructured mesh with refined wake region will then be used. For the case of a Reynolds number of 200, only the fully unstructured mesh will be used for the simulation. Again, results will be compared for the various time stepping routines, however, the conjugate gradient method as the pressure correction solver will now be dropped due to its slow performance. The direct solver to produce a solution to the pressure correction equation will be the only option under consideration.

The initial set of results are presented for the case where $Re = 100$, on the structured mesh produced in the same way as for the inviscid case. The mesh contains 14200 elements and 7200 nodes, with a total for 200 boundary nodes, 100 of these are located around the cylinder. The run-times for this case and the lift and drag coefficients are given in table (6.6). In table (6.6) the Δt for the explicit case is just an indicator as the

	Δt	CPU time	C_L	C_D
Explicit	0.0126	385s	± 0.36	1.405 ± 0.011
Implicit FE	0.02	369s	± 0.46	1.605 ± 0.016
Implicit CN	0.025	370s	± 0.42	1.539 ± 0.013

Table 6.6: CPU times, time step sizes and lift and drag coefficient comparison for flow around a circular cylinder at $Re = 100$, using three time stepping routines.

time step is adaptive for this type of time stepping routine. To keep consistency, the final time step size of the simulation is taken, as vortex shedding has fully developed at this stage and Δt is likely to remain constant.

A quick analysis of the results in table (6.6) show that the implicit case once again struggles to outperform the explicit case. The implicit time step, not being large enough to counteract extra computation time. The lift and drag coefficients vary depending on the time stepping being used, this could possibly be due to the varying time accuracy of the various time stepping schemes.

A finer fully unstructured mesh is also considered. This mesh consists of 101240 elements and 50776 nodes is used. The mesh has a refined wake region to allow better capturing of the vortex shedding, the section of the mesh is given in figure (6.43). Similarly to the coarse mesh case run-times, the lift coefficients and the drag coefficients

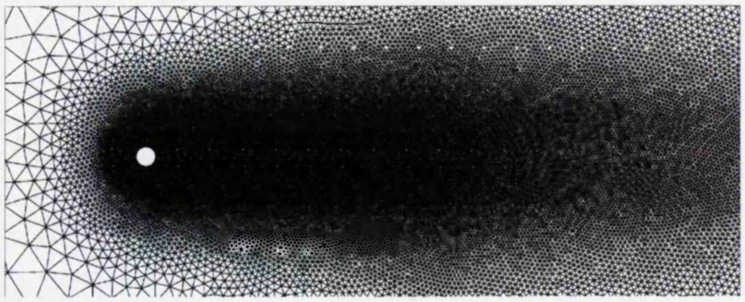


Figure 6.43: Unstructured mesh around a circular cylinder of diameter one, with refined wake region.

are compared for each of the three time stepping routines, see table (6.7).

	Δt	CPU time	C_L	C_D
Explicit	0.0061	1h 33m	± 0.33	1.386 ± 0.0094
Implicit FE	0.015	1h 5m	± 0.4	1.537 ± 0.0133
Implicit CN	0.015	1h 16m	± 0.37	1.464 ± 0.0116

Table 6.7: Comparison of CPU times and lift and drag coefficients for the explicit and two implicit time stepping routines for flow around a circular cylinder with $Re = 100$.

For the flow around a circular cylinder there exist many results in the literature. To validate the algorithm, the lift and drag coefficients obtained here can be compared to the results from the literature, see table (6.8). The effect of varying lift and drag

	C_L	C_D
Morgan et.al. [2]	± 0.32	
Zhang et.al. [104]	± 0.34	1.36 ± 0.01
De Palma et.al. [105]	± 0.331	1.32 ± 0.01
Codina et.al. OSS [106]	± 0.38	1.53 ± 0.0095
Codina et.al. ASGS [106]	± 0.36	1.53 ± 0.01
Codina et.al. CBS [106]	± 0.30	1.485 ± 0.007
Linnick and Fasel [107]	± 0.337	1.38 ± 0.01
Liu et. al. [108]	± 0.339	1.35 ± 0.012
Present coarse mesh explicit	± 0.36	1.405 ± 0.011
Present coarse mesh implicit FE	± 0.46	1.605 ± 0.016
Present coarse mesh inplicit CN	± 0.42	1.539 ± 0.013
Present fine mesh explicit	± 0.33	1.386 ± 0.0094
Present fine mesh implicit FE	± 0.4	1.537 ± 0.0133
Present fine mesh inplicit CN	± 0.37	1.464 ± 0.0116

Table 6.8: Comparison of lift and drag coefficients with literature for flow around a circular cylinder at $Re = 100$

coefficients for the various time stepping routines is once again observed, a similar effect is observed by Codina et.al. [106]. To re-iterate a possible cause of this could be the varying time accuracy.

The semi-implicit time stepping routine was developed to allow the use of stretched meshes. The testing of the algorithm using this test case is provided for completeness and so the varying lift and drag coefficients is an issue that requires further investigation.

As expected it is demonstrated that the larger fully unstructured mesh, when used with the explicit case displays a closer result to most of the literature than the smaller more coarse mesh.

The first unsteady test case presents a good opportunity to compare the unstructured

MAC algorithm to other incompressible flow solvers. The solver considered is the in-house incompressible finite volume solver. The initial run of the in-house solver using the small coarse mesh is compared to the the unstructured MAC solver and the literature [2], see figure (6.44). Only the lift calculated from the pressure force is shown in figure (6.44), the coarse mesh therefore shows an exact comparison to the literature.

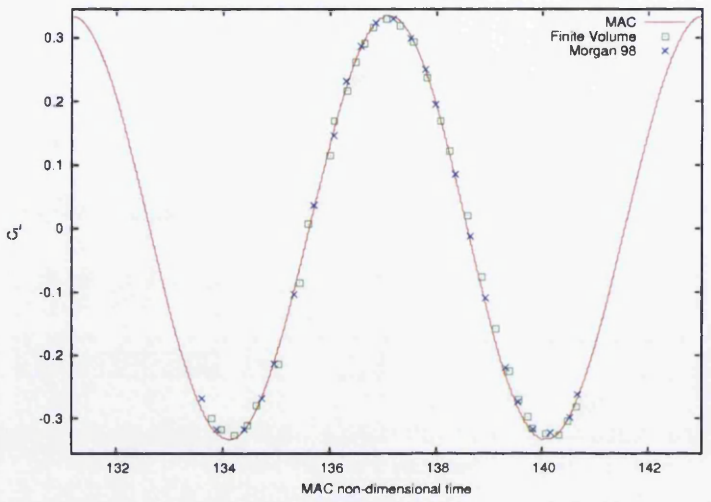
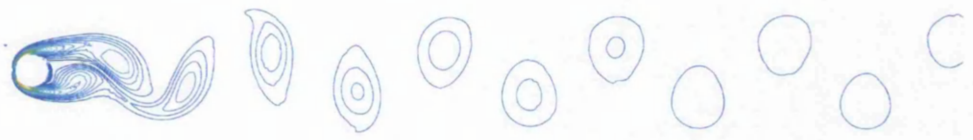


Figure 6.44: Lift coefficient for flow round a circular cylinder at $Re = 100$, comparison of MAC code, finte volume code and literature [2]

Later comparisons using the fine mesh show an under-estimate, therefore the inclusion of the wall shear stress is needed to compare more accurately to the literature. An adverse effect was noticed when the wall shear stress was included using the approximation given in chapter 5. After vortex shedding has fully developed the lift oscillations begin to grow, a similar oscillation is observed in the drag, although the solution still remained stable. The results in table (6.8) are obtained at the uniform point in the lift forces before the adverse affect occurred. When the wall shear stress is not included these oscillations do not occur and uniform oscillatory behaviour is observed. It can also be noted that the finite volume and unstructured MAC algorithm both produce a drag coefficient of around one when only the pressure effects are used to calculate the drag i.e. the shear stresses are not included in the calculation of the drag.

This finite volume simulation of the small coarse mesh required around 6 hours, a substantial amount longer than the MAC code. The finite volume solver produced a successful solution, which was four times slower than the unstructured MAC solver on the same mesh. All comparison were made using the explicit scheme, the semi-implicit solver had not been devised at this time.

It is also of interest to compare the vorticity results for both solvers, when using the fine mesh, see figure (6.45). The MAC solver retains the solution throughout the wake


(a) Vorticity at end of simulation using the unstructured MAC solver.

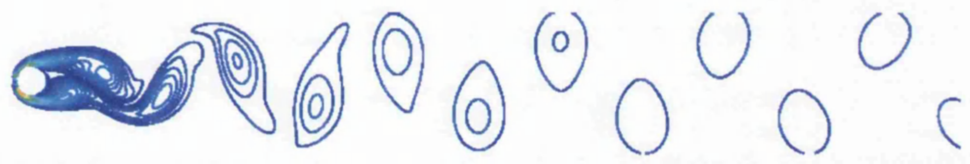

(b) Vorticity at end of simulation using a finite volume solver.

Figure 6.45: Vorticity plots using the unstructured MAC solver and a finite volume solver.

region, whereas the finite volume solver has a diffusing effect on the vorticity as the vortices approach the far field.

Simulations using the fully unstructured mesh are of more interest, they demonstrate that the devised MAC solver runs efficiently and accurately (the solutions are more accurate than the coarse mesh). The wake region is also captured in detail using this mesh and so for the case where $Re = 200$, simulations will only be carried out using the fully unstructured mesh. As before, results and run-times are compared for the various time stepping routines, see table (6.9). To validate the simulation, the lift and drag

	Δt	CPU time	C_L	C_D
Explicit	0.007	2h 10m	± 0.70	1.394 ± 0.0467
Implicit FE	0.015	2h 45m	± 0.75	1.463 ± 0.0512
Implicit CN	0.015	3h 14m	± 0.73	1.430 ± 0.0490

Table 6.9: Comparison of CPU times and lift and drag coefficients for the explicit and two implicit time stepping routines for flow around a circular cylinder with $Re = 200$.

coefficients are compared to the literature, see table (6.10).

Results in the literature are more variable than those for $Re = 100$, the results produced by the unstructured MAC scheme using all the time stepping variants are a reasonable comparison to these results. The explicit time stepping routine appears to produce the best match, particularly to the result of Linnick and Fasel [107]. To demonstrate the development of vortex shedding the vorticity at the end of the simulation using the explicit time stepping scheme is shown in figure (6.46).

	C_L	C_D
Zhang et.al. [104]	± 0.66	1.34 ± 0.03
De Palma et.al. [105]	± 0.68	1.34 ± 0.045
Pan and Damodaran [109]	± 0.63	1.37 ± 0.04
Kiris and Kwak [110]	± 0.67	1.27 ± 0.04
Linnick and Fasel [107]	± 0.70	1.37 ± 0.046
Liu et. al. [108]	± 0.69	1.31 ± 0.049
Belov et.al. [6]	± 0.64	1.10 ± 0.042
Present fine mesh explicit	± 0.70	1.394 ± 0.0467
Present fine mesh implicit FE	± 0.75	1.463 ± 0.0512
Present fine mesh inplicit CN	± 0.73	1.430 ± 0.0490

Table 6.10: Comparison of lift and drag coefficients with the literature for the flow around a circular cylinder at $Re = 200$

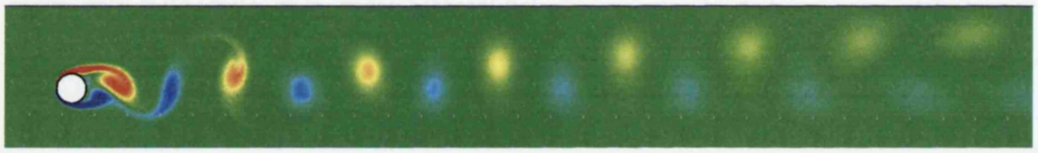


Figure 6.46: Vorticity at the end of the simulation for flow around a circular cylinder at $Re = 200$. Colour scheme is res=high, blue=low.

Oscillations are observed in both the lift and drag coefficient, as for the previous case the values in table (6.10) where taken before these occur.

6.6 Summary

The benchmark tests all confirm that that unstructured MAC algorithm produces stable and accurate solutions when compared to the literature and when good quality meshes are used. The short study on skewed meshes demonstrates that if the Delaunay triangles are obtuse and the circumcentre lies outside then a stable solution is not possible. Using mesh merging, to create a cell centre inside a quadrilateral or moving the circumcentre just inside the triangles (a technique that can create short enough Voronoï edges to allow merging) will allow stable solutions.

The early results with the explicit method demonstrate that the solver speed is highly dependent on the solution technique used for the pressure correction equation. The use of a standard CG method which is not highly optimised is considerably slower than the MA57 direct solver. If an iterative solver were required, alternatives are available and further optimisation may present a faster method. It is important to recognise from this observation that main computational expense of the unstructured MAC method is in the solution of the pressure correction equation. To ensure the unstructured MAC method is efficient an appropriate solution technique needs to be applied to solve the implicit system created from the pressure correction equation discretisations.

The semi-implicit method offers very little benefit on for steady case or the smaller meshes. It only begins to outperform the explicit solver in the unsteady cylinder case on the larger mesh. The semi-implicit method is also using a highly developed direct solver technique so run-times in this case could be considered optimal when using a direct solver approach. Similarly to the solution of the pressure correction equation, the speed of using the semi-implict scheme will be dependent on the efficency of the linear system solver used. It is feasible that an alternative solution technique (perhaps an iterative approach) could be more efficient if it had the same level of development. The alternative technique would have to allow solution of systems with a non-symmetric matrix, this rules out the CG method as an option.

Chapter 7

Further Results

The tests in the previous chapter are standard tests for a CFD solver and validate the results obtained from the unstructured MAC solver, however, they do not stretch the solver to its limits, or even demonstrate its capabilities for more complex geometries. Therefore, simulations around a selection of aerofoils have also been tested, these include the NACA0012, an SD7003 and a multi element aerofoil. There are some results available to some of these problems in the literature and where available the unstructured MAC results have been compared, however, these results are much less well documented and in some cases pose questionable outcomes. The aerofoil simulation really pushed the standard MAC solvers capabilities. It is here that the need for the optimised meshes previously mentioned is really necessary else orthogonality is not retained enough to produce a stable solution. The need for an implicit time stepping scheme will also be demonstrated in the use of stretched meshes to capture viscous effects. This chapter presents the results for; flow over a SD7003 aerofoil; flow over a NACA0012 aerofoil and finally flow over a multi element aerofoil. The last section in this chapter demonstrates a free surface implementation of the unstructured MAC algorithm through the dam break problem. The results for this case are basic but still demonstrates that the free surface capabilities can be included in the algorithm.

7.1 Flow Around SD7003

The previous three examples are good benchmark test problems, to further demonstrate the codes capabilities more complex geometries are required. These more complex geometries take the form of aerofoils, the first of which to be considered is the SD7003 aerofoil [3], the geometry of which is given in figure (7.1). A test case with $Re = 10000$

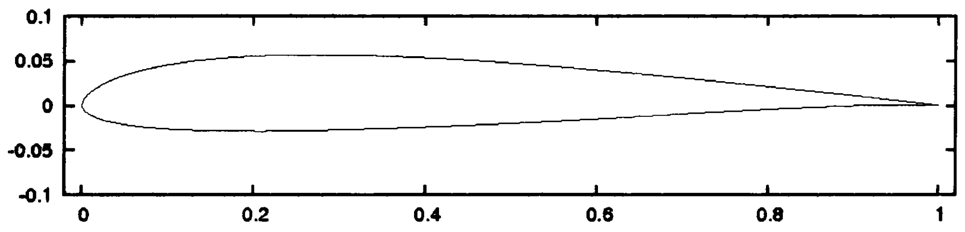
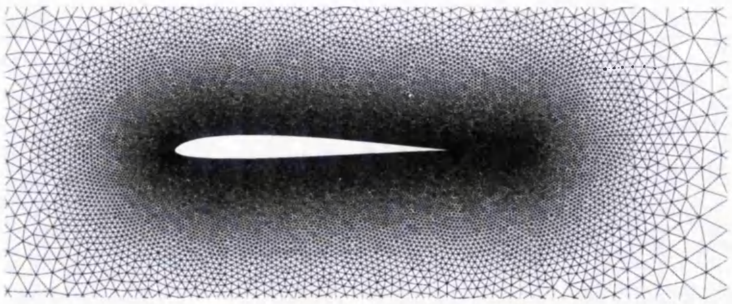


Figure 7.1: Geometry of a SD7003 aerofoil [3]

and an angle of attack of 4^o has been presented by Uranga et.al. on two occasion [111, 112] and also by Castonguay [4]. It is reported that at a Reynolds number of 10000, the flow still exhibits a laminar regime and is still primarily a two dimensional flow. Therefore, the devised unstructured MAC scheme should be able to produce results for this simulation, presenting an opportunity to validate the MAC solver against other numerical solutions for a problem at a high Reynolds number.

The high Reynolds number requires a fine mesh resolution around the aerofoil in order to fully capture the viscous effects. The fine resolution is only required in the normal direction from the aerofoil and so a hybrid mesh can be used. A hybrid mesh posed some new challenges for the code, not only with the implementation to allow the MAC solver to process hybrid meshes but also in retaining the dual orthogonality of the mesh. Hybrid meshing techniques can produce some obtuse elements particularly off the tail end of the aerofoil. These obtuse elements are the bad elements which have a circumcentre outside of the the element and lead to unstable solutions.

The initial hybrid mesh to be used, contains only three stretched boundary layers, see figure (7.2). The devised MAC algorithm using the circumcentre as Delaunay triangle centres will not produce a stable solution using the mesh in figure (7.2). Instead an unstable solution is produced which initially creates large values at the location of the bad elements. Figure (7.3) shows the solution as it goes unstable. The figures show the mesh region at the tail edge of the aerofoil. Initially, to deduce that the bad elements were causing the instability, the mesh elements were corrected by hand so that the circumcentre would lie inside the triangles. After this change a stable solution was possible. Bad elements away from the aerofoil did not need to be corrected. A justification for the bad elements near the aerofoil only being problematic can be understood through the discretisations, in particular through the pressure term. The pressure term discretisation involves a central difference to approximate the gradient. The position the pressure gradient is required at is at the same location as the velocity variable i.e.


(a) Mesh around the SD7003 aerofoil

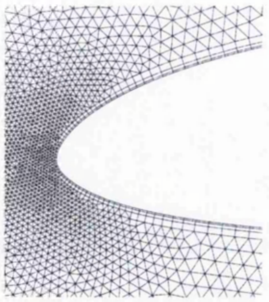
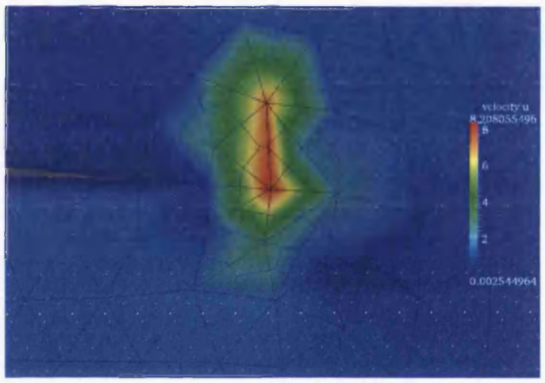

(b) Nose section of mesh for SD7003 aerofoil, three stretched boundary layers are visible

Figure 7.2: Hybrid mesh with three boundary layers for an SD7003 aerofoil


(a) Horizontal velocity

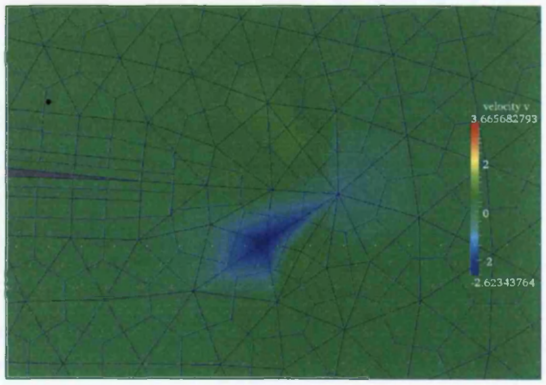

(b) Vertical velocity

Figure 7.3: Unstable solution for flow around an SD7003 aerofoil at trailing edge using the mesh in figure (7.2).

at the midpoint of a Delaunay triangle edge. For bad elements the circumcentre lies outside the cell, because the pressure is stored at the circumcentres the gradient will be approximated to second order at the midpoint between the two circumcentres when using the central difference approximation [9]. This is not the required position (i.e. the same position as the velocity variable). If there is little variation in the pressure field the discrepancy in position is unlikely to have much affect, as all pressure gradients are similar throughout the area. However, for a largely varying pressure field such as those close to the aerofoil, the pressure gradients can vary greatly in space and so position of the gradient is important in relating the correct pressure gradient to the velocity. The loss in accuracy when meshes become non-orthogonal can also be explained from the discretisations and the central difference approximation not being located at the position the velocity is required at. Since the gradient is not approximated at the correct location there will be an error associated with the gradient. In areas where the flow variables change by a large magnitude these errors will be greater.

The unstable solution demonstrates the need for the techniques given in section (4.1). The chosen technique is to optimise the mesh, after-which successful results can be obtained. The lift and drag coefficients obtained when using the optimised three boundary layer mesh are given in figures (7.4,7.5). The lift and drag coefficients given in fig-

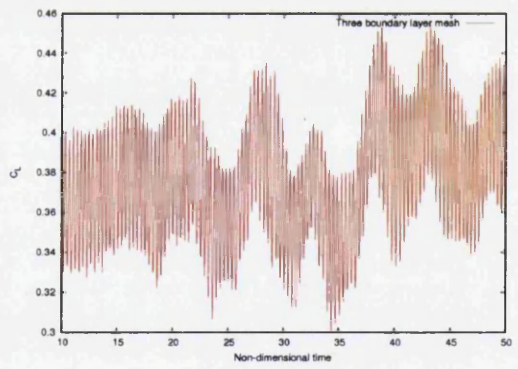


Figure 7.4: Lift coefficient for flow around an SD7003 aerofoil at $Re = 10000$ using an unstructured mesh with three stretched boundary layers

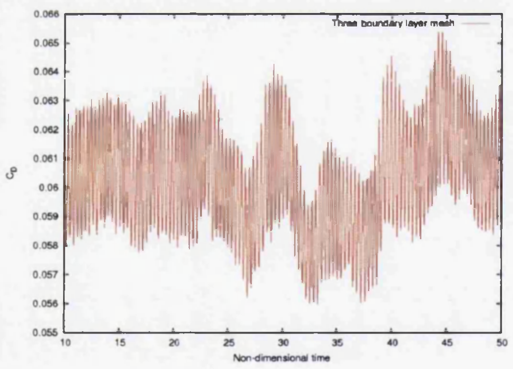
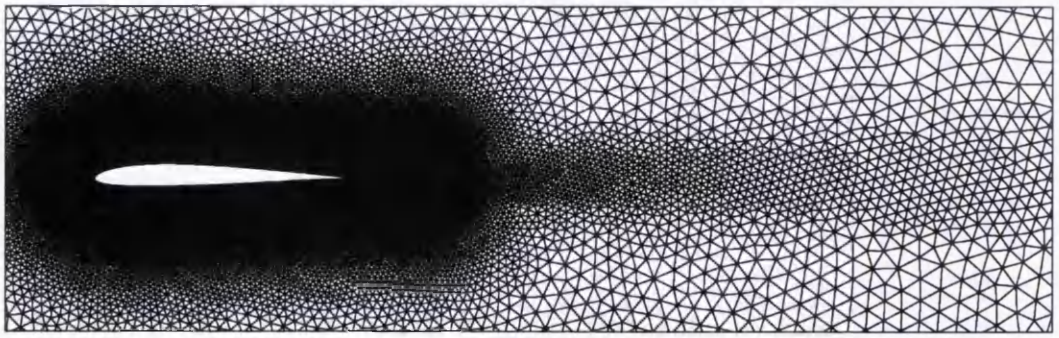


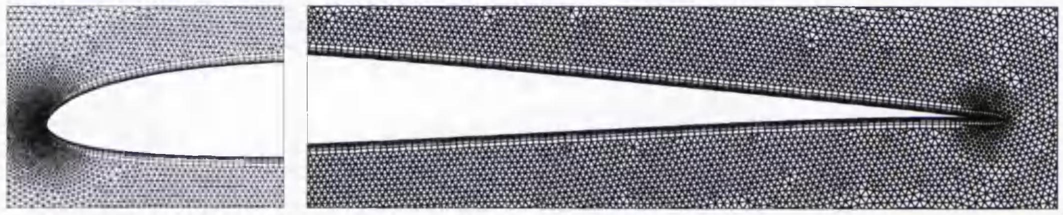
Figure 7.5: Drag coefficient for flow around an SD7003 aerofoil at $Re = 10000$ using an unstructured mesh with three stretched boundary layers

ures (7.4,7.5) are not desirable results, the values oscillations are not around a fixed mean value. A coarse mesh in the boundary region was deemed as the cause of these oscillations. The coarse mesh does not allow the viscous effects to be fully captured around the aerofoil boundary. Therefore, a mesh with a greater number of stretched boundary layers is required, the next set of results demonstrate the use of a mesh such as this.

The next mesh used in an attempt to produce a suitable solution is one consisting of ten stretched boundary layers around the aerofoil, see figure (7.6). The statistics for



(a) SD mesh including wake region



(b) Close up of nose region of SD aerofoil, displaying ten boundary layers

(c) Close up of tail region of SD aerofoil, displaying ten boundary layer

Figure 7.6: Hybrid mesh for an SD7003 aerofoil with refined wake region and ten layers of stretched boundary elements

this mesh are as follows: 4443 quadrilateral elements, 89235 triangular elements and 49363 nodes. The explicit scheme is very slow to produce a solution, a very small time step is required to produce a stable solution when using the explicit scheme. Test runs using the explicit scheme with the mesh in figure 7.6 showed that vortex shedding had not developed after a lengthy CPU time. To combat this issue, the semi-implicit time stepping scheme has been implemented, allowing a much larger time step to be used and a solution exhibiting vortex shedding is obtained in a reasonable amount of time. Although this semi-implicit scheme was implemented purely to allow the use of hybrid stretched meshes, previous results with more standard benchmark problems further validate its use and further test the unstructured MAC solvers capabilities.

The semi-implicit time stepping scheme allowed the use of larger time steps and stable unsteady solutions were produced on the mesh with ten boundary layers after approximately 24 hours of CPU time. There is a noticeable improvement in the lift and drag coefficients over the three boundary layer case, see figures (7.7,7.8). The results although still exhibit minor oscillations about a mean value, however, an average of the

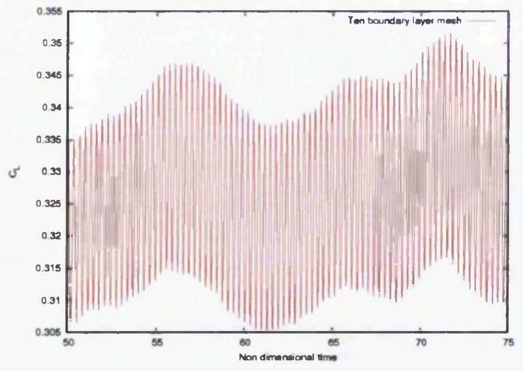


Figure 7.7: Lift coefficient for flow around an SD7003 aerofoil at $Re = 10000$ using an unstructured mesh with ten stretched boundary layers.

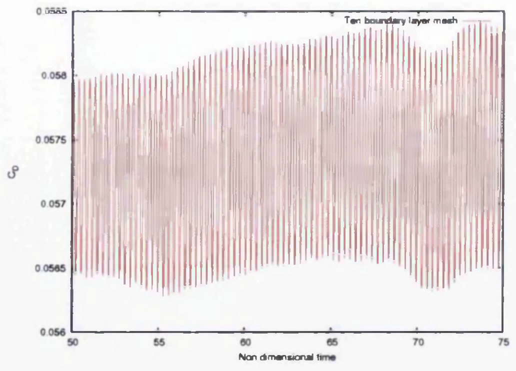


Figure 7.8: Drag coefficient for flow around an SD7003 aerofoil at $Re = 10000$ using an unstructured mesh with ten stretched boundary layers.

lift and drag coefficients is still taken and the results compared to those in the literature, see table (7.1). Table (7.1) shows reasonable results compared to the literature, the lift

	C_L	C_D
Present	0.324	0.0572
Xfoil as in [112]	0.2711	0.04578
Uranga et.al. [112] grid 2, 2D	0.3755	0.04978
Uranga et.al. [111]	0.3826	0.0504
Castonguay et.al. [4]	0.372	0.0492
Galbraith and Visbal as in [4]	0.36	0.047

Table 7.1: Comparison of lift and drag coefficients to literature for the flow around an SD7003 aerofoil at $Re = 10000$.

coefficient is in-between that given by the xfoil code which is used as a comparison in [112] and the other literature results, the drag offers a slightly better comparison. The vorticity at the end of the simulation is given in figure (7.9), the range of values is between -40 and 40 and compares well to the vorticity plot given in [112].

To further validate the results, the pressure coefficient are compared to that given in the literature [4]. For the case of the MAC results the pressure coefficient is that at the end of the solution whereas that in the literature is an average over time, however, the results still demonstrate a reasonable comparison, see figure (7.10). Flow over an SD7003 aerofoil is not an extensively used test problem and only few results exist in the literature. It is therefore difficult to say which results are more accurate. The use of a mesh with more boundary layers may shed further light on the situation but would further decrease the time step. As the ten boundary layer case requires over a day to

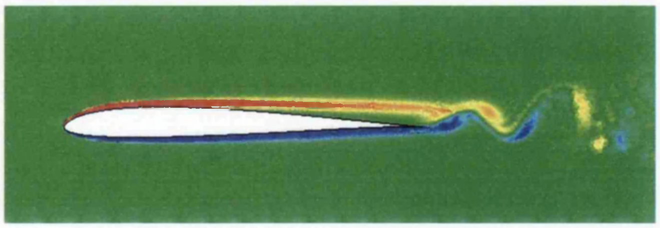


Figure 7.9: Vorticity at the end of the simulation for the flow around an SD7003 aerofoil at $Re = 10000$. Blue=low, red=high.

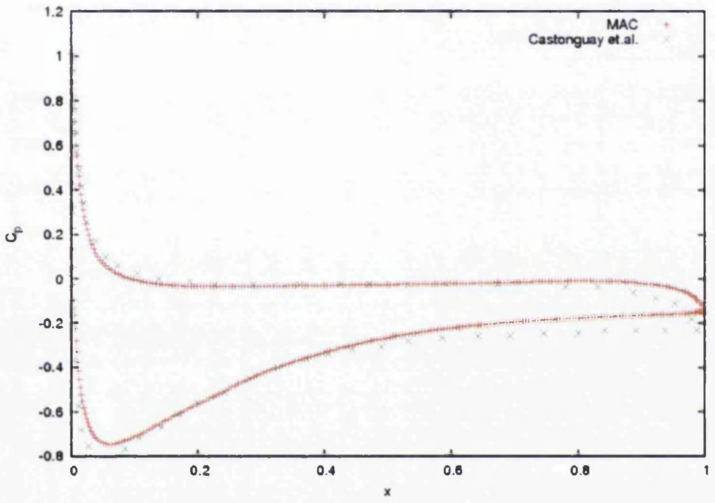


Figure 7.10: Coefficient of pressure comparison to literature [4] for flow around an SD7003 aerofoil at $Re = 10000$

produce the given solution, the introduction of more boundary layers would reduce the time step further, meaning that more computation time is required. It then becomes questionable whether this use of the unstructured MAC algorithm for these types of flow problems is where is it best utilised.

7.2 Flow Around NACA0012

Another aerofoil to consider is the NACA0012 aerofoil [5]. The aerofoil has a symmetric geometry, which can be seen in figure (7.11). In the literature, there exists a test case for an unsteady incompressible flow around a NACA0012 at $Re = 1000$ and with an angle of attack of 20^o [6], thus a comparison is made with the unstructured MAC solver. To capture the vortex shedding well the mesh requires a refined wake region. As the angle of attack is 20^o, the refined region should be at this angle to the aerofoil,

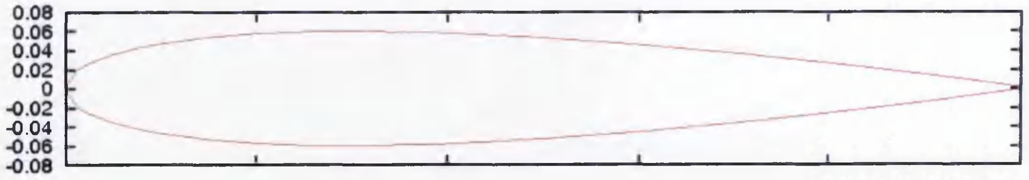


Figure 7.11: Geometry of a NACA0012 aerofoil [5]

see figure (7.12). To allow viscous effects to be fully captured, the boundary region

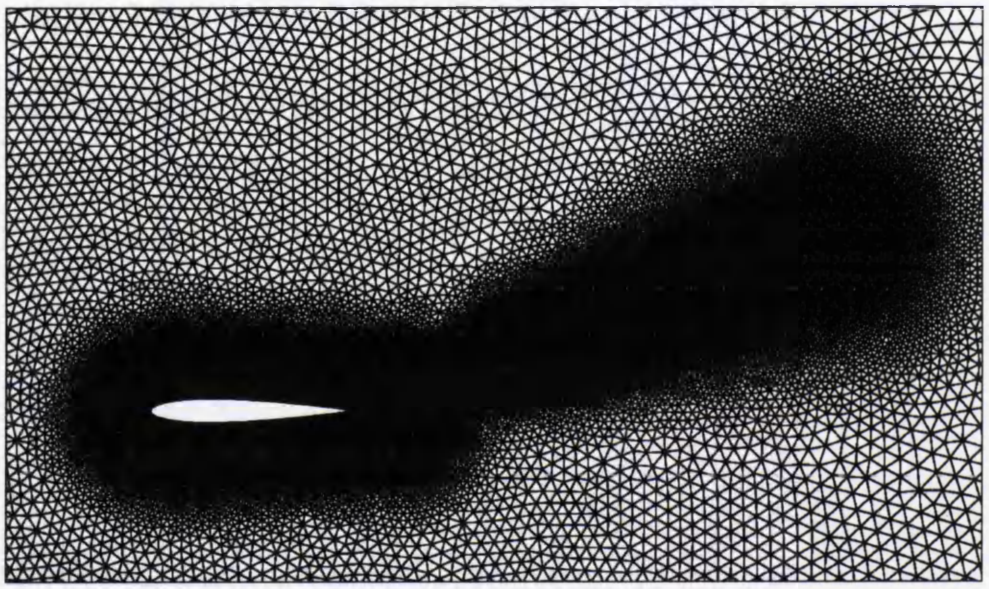


Figure 7.12: Section of a NACA0012 mesh with refined wake region suitable for use with an angle of attack of 20^o.

around the aerofoil has five boundary layers of stretched quadrilateral elements, see figure (7.13). The full statistics for this mesh are: 2182 quadrilateral boundary elements,

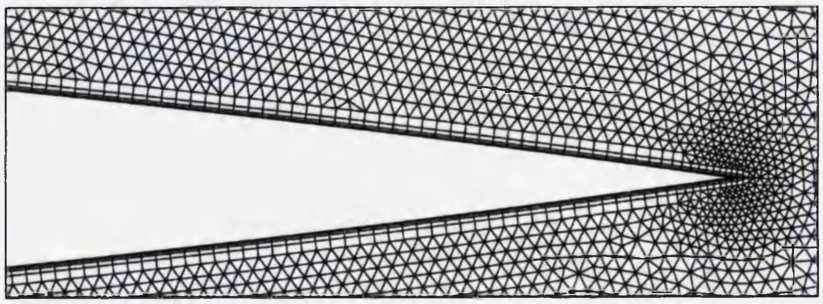


Figure 7.13: Boundary region around the NACA0012 aerofoil, displaying five boundary layers.

121249 triangular elements and 63133 nodes.

The unsteady solution was left to run for 70000 iterations to allow vortex shedding to fully develop. Using an implicit time stepping scheme, a run time of approximately nine hours was achieved on an AMD opteron 240 processor. The lift coefficient compares well to the literature, see figure (7.14).

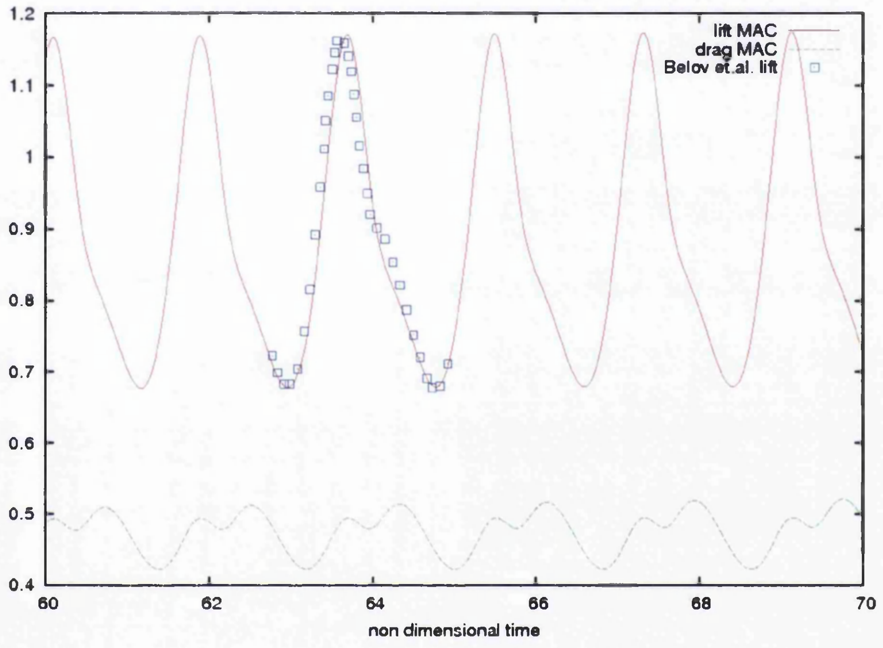


Figure 7.14: Lift and drag coefficients for the flow around a NACA0012 aerofoil at $Re = 1000$ and a 20^o angle of attack. Contains a comparison of the lift coefficient to the literature [6]

The time period in the literature results differs from that of the MAC solver and so one oscillation of the MAC algorithm has been matched to a section of the literature results. The lift coefficient compare reasonably well, the range of values being the same and only a slight discrepancy in the pattern observed in figure (7.14). The drag coefficient does not compare well, its overall values are approximately 0.1 higher than the literature, however, the pattern is similar. This is a complex example, as a lot of shearing stress is produced, the results are promising and as the author of this thesis cannot find this test case in any other article, it is reasonable to accept them. The overall behaviour of the flow is correct when comparing to this particular example. Further comparison to experimental data, or other numerical data could help in determining the issues arising in this test case.

The behaviour of the flow can be demonstrated by looking at the results for each variable at the end of the simulation. Results for the entire domain is shown in figure (7.15). It can be seen that vortex shedding has fully developed and the wake is in the correct direction.

7.3 Flow Around a Multi Element Aerofoil

This is the most complicated shape to be considered. Although there are no results for incompressible flow to compare to, it is still useful to demonstrate that successful simulations can be performed on this type of geometry.

For this test case, a mesh of 166294 elements and 84022 nodes is used, see figures 7.16 and 7.17. To successfully produce a solution, the mesh that required optimisation. A high Reynolds number of 10000 was used along with a 4^o angle of attack, the resulting solution can be seen in figure (7.18).

The results in figure (7.18) were obtained using the explicit time stepping scheme. A simulation of 90000 iterations required around nine hours of CPU time on an AMD opteron processor. The domain requires a large mesh and so the computation time is long, for the explicit time stepping case a time step of order 10^{-6} was required. The test case was also simulated using the semi-implicit scheme, a time step of one order of magnitude larger could and was used, thus a simulation to a similar accuracy took approximately four hours.

The conjugate gradient method as the pressure correction solver was not appropriate for this test case. The results did not look viable, it is possible that the CG method was not converging close enough to the correct solution within the maximum specified iteration. It is in this case that the use of a preconditioner with the CG method may have been able to improve the condition of the solution matrix and the convergence of the solver. These results displayed here where produced using the MA57 solver to solve the pressure correction equation, the solution quality and CPU time were greatly improved in doing so.

7.4 Aerofoil test case summary

Reasonable results can be produced for the aerofoil meshes. As demonstrated in the SD7003 case it is important to remove bad elements in the mesh from areas of high flow gradients to prevent unstable solutions. The dependence on the high quality meshes demonstrates a limitation of the technique. There are overheads in optimising the meshes, so the unstructured MAC technique would be better suited to cases which require one mesh and are run many times e.g. for testing flow patterns with varying Reynolds numbers.

The alternative to improving the mesh quality is to modify the discretisations so that a stable solution on non-orthogonal meshes with bad elements is possible. One method is to transform the discretisations to a generalized curvilinear coordinate systems such as the method applied by Zhu et.al. [113]. Other alternative methods have been developed when using the SIMPLE method with collocated variables. The use of collocated variables also requires an interpolation scheme to handle the checkerboarding pressure effect. The method of Lehnhäuser and Schäfer looks at the discretisation used in the presure correction equation to allow solution on non-orthogonal meshes [114]. In the technique applied by Lebon et.al. a non-orthognality correction is applied to the diffusion term and an interpolation scheme is applied to indvidual problematic elements. The scheme is tested using the steady state lid driven cavity problem on skewed meshes [115]. The work of McBride looks at the storage position of the collocated variables using a finite volume discretisation and the SIMPLE algorithm [116]. The method is tested using the lid driven cavity problem on distorted meshes. A posibility in the unstructured MAC algorithm might be to modify the discretisations so that the central difference approximations can be interpolated back to location of the velocity. This may present a method of improving solutions on distorted meshes without the need for improving the mesh quality.

The aerofoil cases demonstrate the need for the use of the semi-implicit discretisation. Stretched meshes, large variation in mesh element size and high Reynolds numbers mean that the viscous time step becomes small. The semi-implicit method removes this constraint and the extra computational effort required in solving the semi-implicit system is outweighed by the reduction in time steps. As mentioned in the previous chapter the semi implicit system is solved using the MA41 direct solver provided by HSL [86]. This solver is efficient, an alternative solver could be used but to continue to exhibit fast run times the alternative solver code would need to be written efficiently.

The CPU times are all on a single processor, based on a cluster super computer. The unstructured MAC code is not a parellel implementation and only requires the one processor, therefore it can easily run on any home pc, with the clock speed of the processor governing the speed of the simulation. For example, test cases were also run on a home pc containing an intel i5 2400 3.1GHz processor, where run-times on this machine were 2-3 times faster than the opteron. The intel i5 machine has 8GB of ram which was sufficient for the meshes used in the presented test cases.

7.5 Free surface Tracking

This section makes a brief summary of the work done on introducing a free surface to the unstructured MAC method and looks at basic example of the dam break problem. The basic description of the method implemented for free surface tracking and an example are included for completeness.

7.5.1 Implementing a Free Surface into the Unstructured MAC code

In the original Marker and Cell algorithm a free surface can be tracked using marker particles. Marker particles have no mass and no size, they are defined as a point location, in two dimensions this is an x and y coordinate. The location of the marker particles indicates which of the mesh elements contain fluid.

The unstructured MAC algorithm can be adapted for free surface tracking by only solving the discrete equations for elements which contain fluid. Once the steps in the MAC algorithm have been followed and the new velocities have been found, the coordinates of the marker particles can be updated.

To summarise, the following steps must be taken to track a free surface using the unstructured MAC algorithm and marker particles.

1. Solve momentum equation for the normal velocity for edges which form mesh elements that contain fluid.
2. Solve the pressure correction equation for mesh elements that contain fluid.
3. Correct the pressure and velocity for mesh elements that contain fluid.
4. Find the tangential velocities
5. Update the marker particle positions.
6. Identify which mesh elements now contain fluid.
7. Identify which edges need to be solved for.
8. Advance to the next time step.

The marker particle coordinates are update using the following equations,

$$x^{m+1} = x^m + \Delta t w_1 \tag{7.1}$$

$$y^{m+1} = y^m + \Delta t w_2 \tag{7.2}$$

where x is the horizontal coordinate of the marker particle, y is the vertical coordinate of the marker particle, $\boldsymbol{w} = (w_1, w_2)^T$ is the Cartesian velocity vector and Δt is the time step. This Cartesian velocity is assumed to be the element velocity for which the marker particle lies within. The element velocity is calculated by using averaging the velocities at all edges of an element converted to Cartesian coordinates.

The positions of the marker particles are used to flag cells as containing fluid if a marker particle lies within them. Only these cells will be solved for in the MAC solver. Elements are flagged as either empty, containing fluid, or being on the free surface. A free surface is indicated as an element which contains marker particles but a neighbouring element is empty. The edges are then flagged to determine if the velocities on those edges need to be solved. For the edges, special consideration is needed if an edge forms a free surface element. Flags are required to identify if an edge lies between; two full cells; two empty cells; a free surface cell and an empty cell; and two free surface cells. These are required to correctly determine whether a pressure variable exists and to correctly find the Voronoï cell area.

7.5.2 The Dam Break Problem

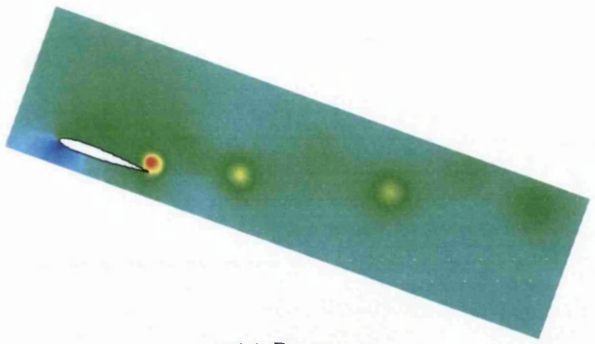
The dam break problem simulates the movement of a body of water after the surface that holds it breaks. To simulate this problem, a gravity forcing is added to the momentum equations to force the body to move downwards. To demonstrate the use of triangular meshes, a mesh of equilateral triangles is used, however this mesh does not line up exactly with the boundaries, see figure (7.19). The mesh consists of a total of 4000 elements. All boundaries have been set to the wall boundary condition. Initially fluid lies within a 1x1 region on the mesh and so this is where the marker particles are placed, see figure (7.20). A total of 6800 marker particles have been used in this example. The mesh extends far enough to the right so that the particles do not interact with the far boundary. The free surface implementation of the unstructured MAC code is then run using a Reynolds number of 10 and a CFL number of 0.5 for 2000 iterations. The marker particles are output every 100 iterations, figure (7.21) show the advancing position of the free surface for various number of iterations. Most of the movement in the free surface is during the earlier iterations.

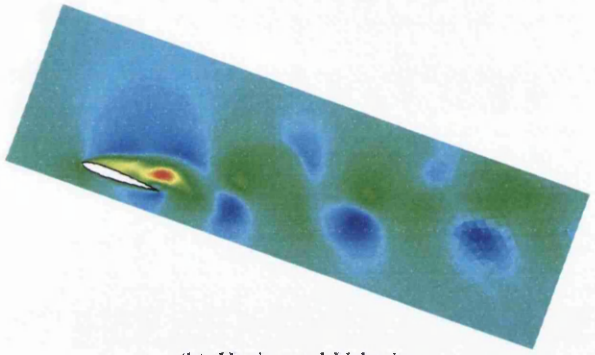
7.5.3 Summary on Free Surface Tracking

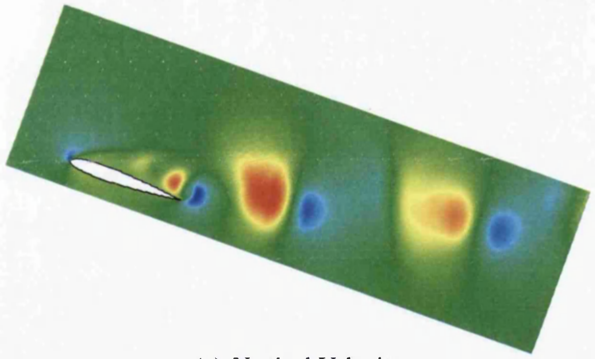
The free surface tracking method implemented uses marker particles. There have been many advances in the field of free surface modelling since the original marker and cell method which use techniques such as level set methods or the volume of fluid method. The results presented here show an initial attempt at modelling the dam break problem.

The results show some success in that the marker particles are advanced through time. The location of the free surface seems reasonable, the particles fall to the bottom and move across, the result that is expected. Further work could be done to validate the results by comparing them to the literature. There are a couple of noticeable issues with the results in figure (7.21), the first being the collection of marker particles which are left at the top of the domain. It is possible that this is either because of the algorithm that places the initial particles, which may have placed them outside the mesh, or because the boundary at the top is a wall boundary and the particles are given a zero velocity. There is also the collection of particles which remain down the side of the domain after 2000 iterations. This is most likely because of the mesh as it does not line up with the boundary. It could also be a result of the wall boundary condition. Future work would determine if this is the correct boundary condition to apply. The model does not move the particle along the bottom of the domain very quickly either, a theory for this is the incorrect representation of the marker particle velocity.

To summarise, a first attempt at modelling a free surface was made during the development of the marker and cell algorithm. Limited success was shown in the implementation of a free surface tracking system. The difficulty in implementing more advanced techniques such as the volume of fluid method, coupled with the need to add the capability for modelling highly viscous flow meant free surface capabilities were not developed further during this project.


(a) Pressure


(b) Horizontal Velocity


(c) Vertical Velocity

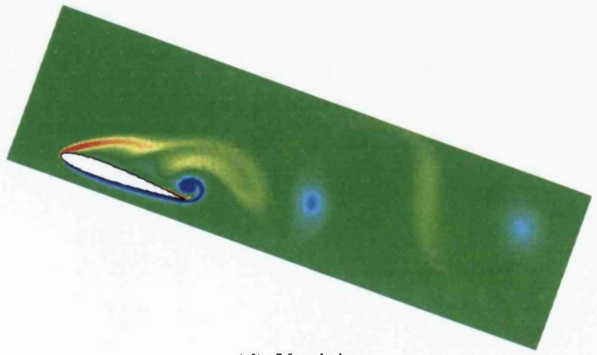

(d) Vorticity

Figure 7.15: Results for the flow around a NACA0012 aerofoil at $Re = 1000$ and a 20^o angle of attack. Blue=low, red=high.

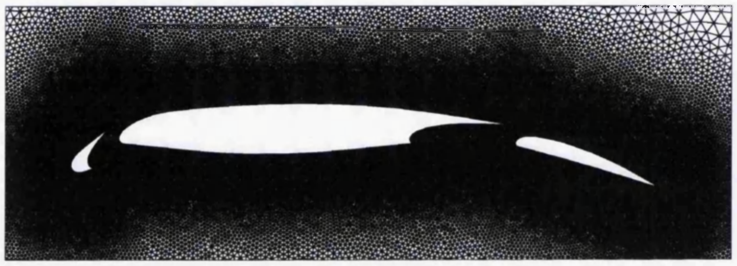
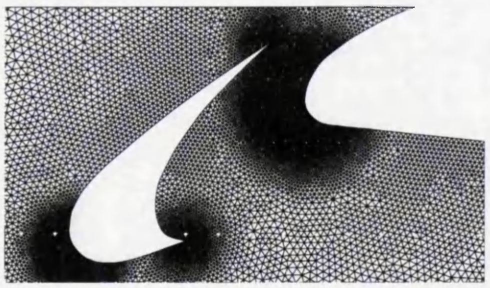
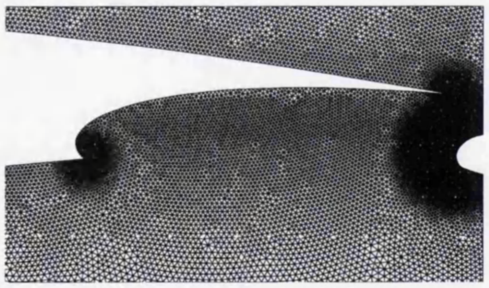


Figure 7.16: Segment of mesh around a multi element aerofoil

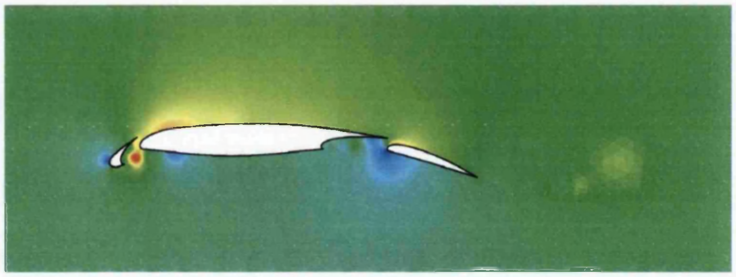


(a) Zoom of nose region of the main largest element and front element of the aerofoil

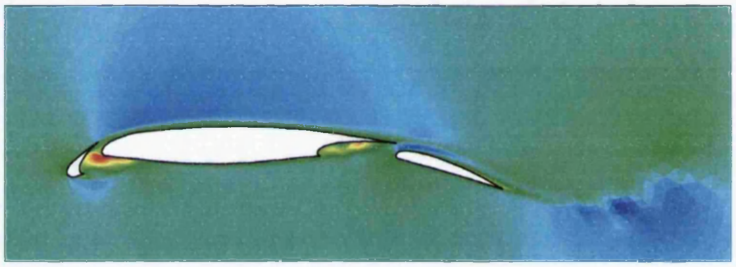


(b) Zoom of tail of the main largest element of the multi element aerofoil

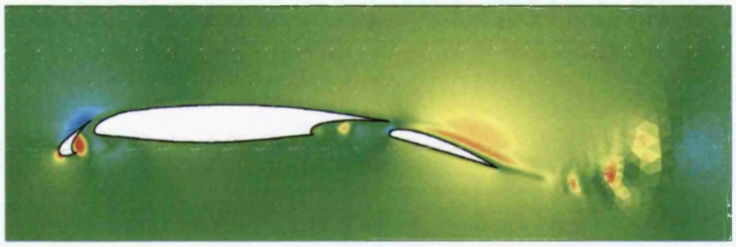
Figure 7.17: Zoomed images of the multi element aerofoil to display the mesh refinement in the boundary regions



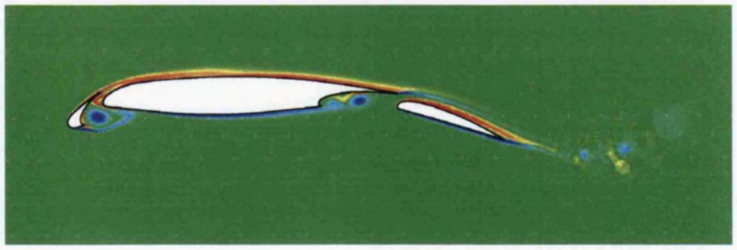
(a) Pressure



(b) Horizontal velocity component



(c) Vertical velocity component



(d) Vorticity

Figure 7.18: Results for the multi element test case with $Re = 10000$ and an angle of attack of 4^o. Blue=low, red=high.

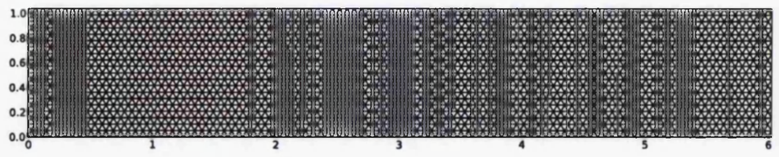


Figure 7.19: Mesh used for modelling the dam break problem

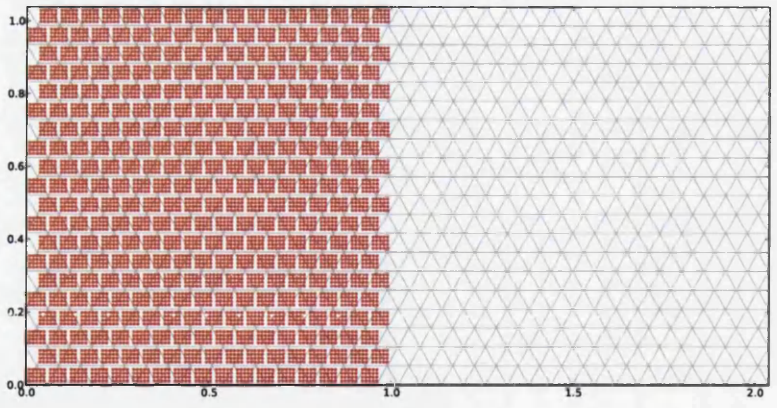
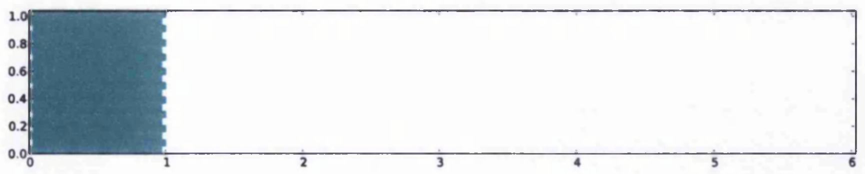
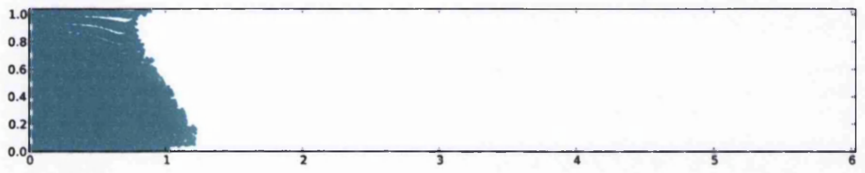


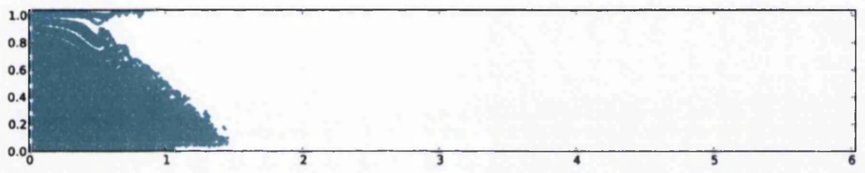
Figure 7.20: Initial position of the marker particles



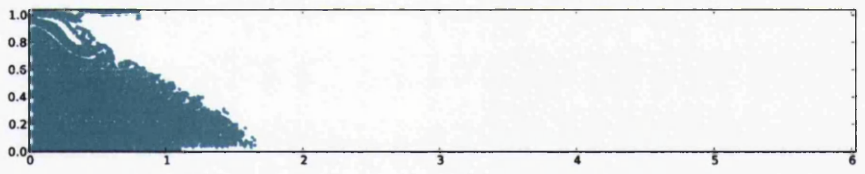
(a) Initial position of the marker particles



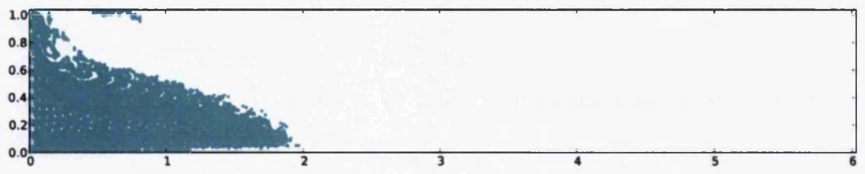
(b) Position of the marker particles after 200 iterations



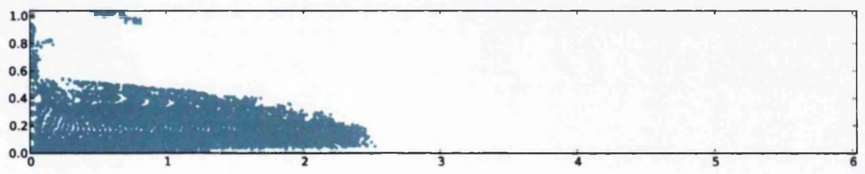
(c) Position of the marker particles after 400 iterations



(d) Position of the marker particles after 600 iterations



(e) Position of the marker particles after 1000 iterations



(f) Position of the marker particles after 2000 iterations

Figure 7.21: Tracking of a free surface using marker particles for the breaking dam test problem

Chapter 8

Conclusion

A novel efficient solution method for incompressible flow has been devised, tested and validated. The marker and cell method is an old technique for Cartesian meshes [12]. Its algorithm and staggered mesh layout in both a Cartesian and triangular framework have been previously researched in the literature. However, the coupling of the marker and cell algorithm with with an unstructured staggered mesh is a topic that has not been found by this author in the literature. The MAC staggered mesh layout has in the past been used but has been combined with other solution methods [57]. This thesis has applied the discretisation for the unstructured MAC mesh to use with the MAC algorithm in an attempt to produce an efficient CFD solver. Comparison of the resulting algorithm with the in–house incompressible flow solvers shows a staggering decrease in the required CPU time. Although it is clear that this run time is goverened by the choice of linear system solver for either the pressure correction equation or when using the semi-implicit discretisations.

Development of the method was not without its problems, many of which were due to the reliance on the existence of a dual orthogonal mesh. Problems were eradicated one by one as they occurred through program testing, the initial hurdle of course was the accurate representation of the discretisations efficiently in Fortran 90 code. When unsteady flow cases were attempted, issues arose in the correct determination of boundary conditions. It was the use of more complex geometries that really pushed the code, many problems were discovered at this stage, including the need for hybrid meshes, the high dependence on dual orthogonal meshes.

A summary of the various stages of the unstructured MAC solver and the purpose of their development is now presented:

- *Explicit Solver:* The explicit solver is the most robust piece of code developed. A high degree of testing was carried out using this code, with many errors and problems being discovered as a result. At each stage problems where identified and solutions were found, starting from the correct representation of boundary conditions to the high dependency on dual orthogonal meshes. The limitation of the explicit scheme is its use with hybrid stretched meshes, due to the quadrilateral elements having short edges and the huge restriction placed on the time step size. This restriction cannot be solved by element merging and so the semi-implicit discretisation has been developed. A further restriction on the solver which is not limited to the explicit time stepping case, is the mesh quality. Meshes that are close to fulfilling the dual orthogonal quality are needed, not only to keep the solution accurate but in some cases to produce a stable solution as demonstrated by the SD7003 test case. In this test case if bad elements are located next to the aerofoil, i.e. in a area of high flow activity, then a stable solution was not possible.

- *Inviscid Scheme:* Inviscid flows can be simulated by not calculating the viscous term and removing the viscous time step. The boundary conditions must also be altered to allow for an inviscid wall. The inviscid scheme is only tested via one example which shows very successful results, a solution very close to the analytical is observed.

- *Pressure Correction Equation Solvers:* Solving the pressure correction equation requires the most computational effort out of all the stages in the MAC algorithm. The results comparing the use of a conjugate gradient solver compared to the direct solver demonstrate this is the case.

- *Hybrid Meshes:* The unstructured MAC solver has the capability for processing hybrid meshes. A desirable quality for the solver as it makes the simulation of viscous flows possible.

- *Semi-Implicit Discretisation:* The semi-implicit discretisation was the final stage of the project, where testing is not as complete as for the explicit case, but successful results are observed. The semi-implicit time stepping scheme was developed to solve the problem with highly stretched meshes in the explicit scheme. The approach worked, making it possible to use time steps of an order of magnitude larger for certain test cases. The implicit scheme does not perform as well on the more 'simple' test cases. The lack of speed improvement is possibly due to convective time step size being more dominant than the viscous time step. In this case the explicit scheme will outperform the semi-implicit scheme as there are no extra overheads in solving the system. The explicit scheme also allows adaptive time stepping, as the minimum possible time step may change from iteration to iteration. In the afore mentioned test cases, it is possible that larger time steps maybe used than those calculated at the beginning of an implicit solution. The purpose of introducing implicit time stepping was to allow the solution of problems using stretched meshes. This goal was achieved and the results for the SD7003 test case compared reasonably well to the literature. The development of semi-implicit time stepping occurred late on in the research and so further testing is still required. The range of results in the cylinder test case identifying an area where further investigation is required, although these could be interpreted as the effect of varying time accuracy of the various time stepping schemes.

- *Dual orthogonal meshes:* The creation of meshes is not within the scope of this thesis, however, for completeness, meshing techniques are discussed. A comment should be made on the need for dual orthogonal meshes, testing of the MAC algorithm revealed how important this was. The main cause of instability of the algorithm was identified by the author as being the presence of obtuse triangles, defined here as bad elements. Crude fixes for these bad elements were implemented into the MAC solver, however, it is the work of Walton [72] that allowed the used of his modified cuckoo search [99] to optimise general meshes.

- *Free surface tracking:* An experimental free surface tracking approach was implemented using marker particles. Initial results demonstrate the tracking of a moving surface but these are not validated and more complex problems have not been tested.

Testing of the MAC solver revealed the cases in which it performs best. For small regular domains, the recorded run times are short, particularly when the MA57 direct solver is used to solve the pressure correction equation. Even in this small test cases the comparison with the CG solver demonstrates the need for an efficient linear solver for the pressure correction equation. It is possible that an iterative solver with more optimised code could be more comparable to the direct solver.

A problem arose when trying to simulate high Reynolds number flows using more complex geometries, where providing a suitable mesh was a major difficulty. Mesh optimisation techniques presented a way forward and allowed the use of highly stretched elements around submerged bodies. As demonstrated for the SD7003 case, the use of stretched elements around the aerofoil allowed better capturing of the lift and drag forces. Results in cases using the stretched meshes showed a huge slow down in the method due to the short mesh edges, creating very small time steps. This was later solved by the use of a semi-implicit scheme, which as a result, sped up some previous simulations. The speed up is again governed by the choice of linear solver, although multiple were not tested, to get the best run times an efficient solver for the semi-implcit system needs to be implemented. It is possible that even in the slow stretched meshes cases that if an efficent solver is not used then the semi-implicit method will still be slower that the explicit scheme.

Using the semi-implcit scheme with the MA41 solver to solve the system did not always improve the run times when compared to the explicit scheme. This is usually the case for the test problems on more regular meshes. Therefore it can be concluded that the choice of time stepping scheme is problem dependent.

Overall, the method shows promise, but its reliance on a dual orthogonal mesh presents a major limitation. The method is also only between 1st and 2nd order accurate due to the central difference discretisations, thus, the method perhaps presents its best application when fast simulations are required many times for one chosen problem. In this case, a working mesh can be created and a problem re-run using the same mesh. High accuracy is not always desired especially when only an indicator on the fluids behaviour is required. Its use for problems that require a higher order of accuracy my not be ideal unless methods that introduce higher order accuracy can be implemented.

Future development of the unstructured MAC method depends on the desired application, if environmental flows are being considered perhaps free surface implementation will be beneficial. If aerospace applications are being considered then reducing the dependency on the mesh and improving the accuracy would be ideal. Of course there are many improvements that could be applied to both. Things to consider are as follows:

- *Further investigation of the semi-implicit scheme:* Highlighted in the results are areas where the semi-implicit scheme does not perform as expected. These results required further investigation to ensure a robust semi-implicit solver.

- *Three Dimensions:* The obvious extension with the work is to add the third dimension. The problems with the meshing in two dimensions meant that this project did not move to three dimensions. Instead, testing and validation of the two dimensional solver to obtain a more robust algorithm was undertaken. The discretisations in three dimensions are not perceived to be the difficult element, Cavendish et.al. [65] has devised the three dimensional discretisations on the Delauany tetrahedral mesh. The difficulty envisaged with the the unstructured MAC algorithm in three dimensions is in producing a suitable dual orthogonal mesh. The work by Xie [98] using Voronoï weighting to produce these types of meshes and the work by Walton [72] has been extended to three dimensional meshes, so optimisation of the mesh should be possible.

- *Compressible Flow:* The use of the MAC discretisations could be investigated for use with compressible flow. Identifying whether the same computational savings can be achieved, there would of course need to be further alteration to the MAC algorithm.

- *Turbulence Model:* To simulate turbulence flows, a turbulence model could be added to the code.

- *Free surface:* Free surface flows have already been experimented with. The difficulty with them is in correctly capturing the position of the free surface. The Volume of Fluid method [50] is one of the most widely used, however, it is mostly used with Cartesian grids and so the sophisticated interface capturing techniques are designed with that in mind. However there do exist interface capturing techniques for unstructured grids, some based on those developed for Cartesian grids [53]. Others combine the VOF method with other techniques such as the level set method [55]

- *Parallelisation:* To further improve the speed of the algorithm, the code could be parallelised. An investigation would need to be carried out on the best possible way of achieving this, whether the domain being split or whether there are certain sections of the code that can be run at the same time.

- *Pressure Correction Equation Solvers:* An investigation could be carried out into the best solver for the pressure correction equation. As it stands, only the conjugate gradient method and a very efficient direct solver have been implemented. Since the direct solver has been highly developed [86], it likely that others will perform to a similar degree of efficiency. There are a vast number of iterative schemes that perform better than the conjugate gradient method, the disadvantage however, being that the solution will not be exact. There is also the consideration of what method will be most efficient in three dimensions, if that avenue was to be investigated.

- *Applications:* This focus of this thesis has been the development of the method rather than the applications it is best suited to. Future work could used the method for simulation of real engineering problems, the results of which would further validate any benefits of the unstructured MAC algorithm over other incompressible fluid solvers.

To summarise, the unstructured MAC method demonstrates potential. Areas for further development have been highlighted which further the capabilities of the unstructured MAC method.

Bibliography

[1] U. Ghia, K. N. Ghia, and C. T. Shin. High-re solutions for incompressible flow using the navier stokes equations and a multigrid method. *Journal of Computational Physics*, 48:387–411, 1982.

[2] K. Morgan and J. Peraire. Unstructured grid finite element methods for fluid mechanics. *Reports on Progress in Physics*, 61:569–638, 1998.

[3] UIUC Airfoil Data Site, http://www.ae.illinois.edu/m-selig/index.html.

[4] P. Castonguay, C. Liang, and A. Jameson. Simulation of transitional flow over airfoils using the spectral difference method. In *AIAA 2010-4626*, 40th Fluids Dynamics Conference and Exhibit, Chicago, Illinois, 2010.

[5] I. H. Abbott and A. E. Von Doenhoff. *Theory of Wing Sections*. Dover, 1959.

[6] A. Belov, L. Martinelli, and A. Jameson. A new implicit algorithm with multigrid for unsteady incompressible flow calculations. *AIAA paper 95-0049*, 1995.

[7] S. W. Kim and T. J. Benson. Comparison of the smac, piso and iterative time-advancing schemes for unsteady flow. *Computers and Fluids*, 21:435–454, 1992.

[8] G. K. Batchelor. *Fluid Dynamics*. Combridge University Press, 1967.

[9] J. A. Anderson. *Computational Fluid Dynamics: The Basics with Applications*. McGraw-Hill International Editions, 1995.

[10] J. .H. Ferziger and M. Peric. *Computational Methods for Fluid Dynamics*. Springer, 2002.

[11] H. K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics The Finite Volume Method*. Pearson, Prentice Hall, 2007.

[12] F.H.Harlow and J.E.Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The Physics of Fluids*, 8:2182–2189, 1965.

[13] M. F. Tome and S. McKee. Gensmac: A computational maker and cell method for free surface flows in general domains. *Journal of Computational Physics*, 110:171–186, 1994.

[14] H. Miyata, S. Nishimura, and A. Masuko. Finite difference simulation of nonliner waves generated by ships of arbitrary three-dimensional configuration. *Journal of Computational Physics*, 60:391–436, 1985.

[15] S. V. Patankar. *Numerical Heat Transfer and Fluid Flow*. Hemisphere, 1980.

[16] P. J. Roache. *Computational Fluid Dynamics*. N.M: Hermosa Publishers, 1972.

[17] N. P. Weatherill. Delaunay triangulation in computational fluid dynamics. *Computers and Mathematics with Applications*, 24:129–150, 1992.

[18] M. E. Braaten and W. Shyy. Comparison of iterative and direct solution methods for viscous flow calculations in body-fitted co-ordinates. *International Journal for Numerical Methods in Fluids*, 6:325–349, 1986.

[19] B. L. Lapworth. Examination of the pressure oscillations arising in the computation of cascade flow using a boundary-fitted co-oredinate system. *International Journal for Numerical Methods in Fluids*, 8:387–404, 1988.

[20] T. Ikohagi and B. R. Shin. Finite-difference schemes for steady incompressible navier-stokes equations in general curvilinear coordinates. *Computers and Fluids*, 19:479–488, 1991.

[21] P. P. Patil and S.Tiwari. Computation of flow past complex geometries using mac algorithm on body-fitted coordinates. *Engineering Applications of Computational Fluid Mechanics*, 3:15–27, 2009.

[22] J. F. Thompson, B. Soni, and N. Weatherill. *Handbook of Grid Generation*. CRC Press LLC, 1999.

[23] J. Peraire, J. Peiro, and K. Morgan. *Advancing Front Grid Generation*, chapter 17, pages 17–1 – 17–22. CRC, 1999.

[24] I. Sazonov, D. Wang, O. Hassan, and K. Morgan. A stitching method for the generation of unstructured meshes for use with co-volume solution techniques. *Computer Methods in Applied Mechanics and Engineering*, 195:1826–1845, 2006.

[25] L. Cheng and S. Armfield. A simplified marker and cell method for unsteady flows on non-staggered grids. *International Journal for Numerical Methods in Fluids*, 21:15–34, 1995.

[26] T. M. Shih, C. H. Tan, and B. C. Hwang. Effects of grid staggering on numerical schemes. *International Journal of Numerical Methods in Fluids*, 9:193–212, 1989.

[27] P. M. Gresho and R. L. Sani. *Incomressible Flow and the Finite Element Method: Volume Two, Isothermal Laminar Flow*. Wiley, 2000.

[28] A. J. Chorin. Numerical solution of the navier-stokes equations. *Mathematics of Computation*, 22:745–762, 1968.

[29] J. Peraire, M. Vahati, K. Morgan, and O. C. Zienkiewicz. Adaptive remeshing for compressible flow computations. *Journal of Computational Physics*, 72:449–466, 1987.

[30] K. Mason, O. Hassan, and K. Morgan. Selective use of higher-order reconstruction within an edge-based finite volume scheme for aerodynamic computations. In *Proceedings of the 16th International Conference on Finite Elements in Flow Problems, March 23-25, Munich, Germany*, 2011.

[31] C. E. Scheidegger, J. L. D. Comba, and R. D. de Cunha. Practical cfd simulations on programmable graphics hardware using smac. *Computer Graphics Forum*, 4:715–728, 2005.

[32] A. J. Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of Computational Physics*, 135:118–125, 1997.

[33] Y. Kallinderis and H. T. Ahn. Incompressible navier-stokes method with general hybrid meshes. *Journal of Computational Physics*, 210:75–108, 2005.

[34] A. Jameson. Time dependent calculations using mulitgrid, with applications to unsteady flows past airfoils and wings. *AIAA paper 951-1596*, 1995.

[35] P. Gresho. On the theory of semi-implicit projection methods for viscous incompressible flow and its implementation via a finite element method that also introduces a nearly consistent mass matrix. part 1: Theory. *International Journal of Numerical Methods for Numerical Methods in Fluids*, 11:587–620, 1990.

[36] M. F. Tome, B. Duffy, and S. McKee. A numerical technique for solving unsteady non-newtonian free surface flows. *Journal of Non-Newtonian Fluid Mechanics*, 62:9–34, 1996.

[37] A. Amsden and F. Harlow. A simplified mac technique for incompressible fluid flow calculations. *Journal of Computational Physics*, 6:322–325, 1970.

[38] S. McKee and Et. Al. The mac method. *Computers and Fluids*, 37:907–930, 2008.

[39] M. O. Deville. Numerical experiments on the mac code for a slow flow. *Journal of Computational Physics*, 15:362–374, 1974.

[40] W. E. Pracht. A numerical method for calculating transient creep flows. *Journal of Computer Physics*, 7:46–60, 1971.

[41] V. Armenio. An improved mac method (simac) for unsteady high-reynolds free surface flows. *International Journal for Numerical Methods in Fluids*, 24:185–214, 1997.

[42] M. F. Tome, A. C. Filho, J. A. Cuminato, N. Mangiavacchi, and S. McKee. Gensmac3d: a numerical method for solving unsteady three-dimensional free surface flows. *International Journal for Numerical Methods in Fluids*, 37:747–796, 2001.

[43] M. F. Tome, N. Mangiavacchi, J. A. Cuminato, A. Castelo, and S. McKee. A finite difference technique for simulating unsteady viscoelastic free surface flows. *Journal of Non-Newtonian Fluid Mechanics*, 106:61–106, 2002.

[44] S. McKee, M .F. Tome, J. A. Cuminato, A. Castelo, and V. G. Ferreira. Recent advances in the marker and cell method. *Archives of Computational Method in Engineering*, 11:107–142, 2004.

[45] C. M. Oishi, J. A. Cuminato, J. Y. Yuan, and S. McKee. Stability of numerical schemes on staggered grids. *Numerical Linear Algebra with Applications*, 15:945–967, 2008.

[46] C. M. Oishi, J. A. Cuminato, V. G. Ferreira, M. F. Tome, and A. Castelo. A stable seni-implicit method for free surface flows. *Journal of Applied Mechanics*, 73:940–948, 2006.

[47] C. M. Oishi, J. A. Cuminato, V. G. Ferreira, A. Castelo, and M. F. Tome. Implementing implcit schemes in gensmac. *TEMA*, 5:259–268, 2004.

[48] B. D. Nichols and C. W. Hirt. Improved free surface boundary conditions for numerical incompressible-flow calculations. *Journal of Computational Physics*, 8:434–448, 1971.

[49] H. Miyata. Finite difference simulation of breaking waves. *Journal of Computational Physics*, 65:179–214, 1986.

[50] C. W. Hirt and B. D. Nichols. Volume of fluid (vof) method for the dynamics of free boundaries. *Journal of Computational Physics*, 39:201–225, 1981.

[51] O. Ubbink and R. I. Issa. A method for capturing sharp fluid interfaces on arbitrary meshes. *Journal of Computational Physics*, 153:26–50, 1999.

[52] W. J. Rider and D. B. Kothe. Reconstructing volume tracking. *Journal of Computational Physics*, 141:112–152, 1998.

[53] M. Huang, B. Chen, and L. Wu. A slic-vof method based on unstructured grids. *Mircogravity Sci. Technol.*, 22:305–314, 2010.

[54] S. Osher and R. O. Fedkiw. Level set methods: An overview and some recent results. *Journal of Computational Physics*, 169:463–502, 2001.

[55] X. Lv, Q. Zou, Y. Zhao, and D. Reeve. A novel coupled level set and volume of fluid method for sharp interface capturing on 3d tetrahedral grids. *Journal of Computational Physics*, 229:2573–2604, 2010.

[56] H. Y. Yoon, I. K. Park, Y. J. Lee, and J. J. Jeong. An unstructured smac algorithm for thermal non-equilibrium two-phase flows. *International Communications in Heat and Mass Transfer*, 36:16–24, 2009.

[57] R. A. Nicolaides, T. A. Porsching, and C. A. Hall. Covolume methods in computational fluid dynamics. In *Computational Fluid Dynamics Review*, pages 279–299. John Wiley, New York, 1995.

[58] R. A. Nicolaides. Flow discretization by complementary volume techniques. In *Proc. 9th AIAA CFD Meeting*, volume AIAA Paper 89-1978, 1989.

[59] R. A. Nicolaides. Direct discretization of planar div-curl problems. *SIAM Journal of Numerical Analysis*, 1:32–56, 1992.

[60] R. Nicolaides and D. Wang. A higer order covolume method for planar div-curl problems. *International Journal for Numerical Methods in Fluids*, 31:299–308, 1999.

[61] S. Choudhury and R. A. Nicolaides. Discretization of incompressible vorticity-velocity equations on triangular meshes. *International Journal for Numerical Methods in Fluids*, 11:823–833, 1990.

[62] R. A. Nicolaides. Analysis and convergence of the mac scheme 1. the linear problem. *SIAM Journal of Numerical Analysis*, 29:1579–1591, 1992.

[63] R. A. Nicolaides and X. Wu. Analysis and convergence of the mac scheme 2. navier-stokes equations. *Mathematics of Computation*, 65:29–44, 1996.

[64] C. A Hall, J. C. Cavendish, and W. H. Frey. The dual variable method for solving fluid flow difference equations on delaunay triangles. *Computers and Fluids*, 20:145–164, 1991.

[65] J. C. Cavendish, C. A. Hall, and T. A. Porsching. A complementary volume approach for modelling three-dimensional navier-stokes equations using dual delaunay/voronoi tessellations. In *The Mathematics of Finite Elements and Applications*, pages 255–266. John WIley & Sons Ltd, 1994.

[66] C. A. Hall, T. A. Porsching, and G. L. Mesina. On a network method for unsteady incompressible fluid flow. *International Journal for Numerical Methods in Fluids*, 15:1383–1406, 1992.

[67] B. Perot. Conservation properties of unstructured staggered mesh schemes. *Journal of Computational Physics*, 159:58–89, 2000.

[68] D. Vidovic, A. Segal, and P. Wesseling. A superlinearly convergent finite volume method for the incompressible navier-stokes equations on staggered unstructured grids. *Journal of Computational Physics*, 198:159–177, 2004.

[69] I. Wenneker, A. Segal, and P. Wesseling. Conservation properties of a new unstructured staggered scheme. *Computers and Fluids*, 32:139–147, 2003.

[70] X. Zhang, D. Schmidt, and B. Perot. Accuracy and conservation properties of a three-dimensional unstructured staggered mesh scheme for fluid dynamics. *Journal of Computational Physics*, 175:764–791, 2002.

[71] B. Perot and R. Nallapati. A moving unstructured staggered mesh method for the simulation of incompressible free-surface flows. *Journal of Computational Physics*, 184:192–214, 2003.

[72] S. Walton, O. Hassan, and K. Morgan. Reduced order mesh optimisation using proper orthogonal decomposition and modified cuckoo search. *Article in production.* to be submitted.

[73] R. Pritchard, O. Hassan, and K. Morgan. A computational method for the simulation of incompressible flow on triangular meshes. In *proceedings of the 18th Annual Conference of the Association of Computational Mechanics in Engineering, University of Southampton, 29-31 March*, 2010.

[74] R. Pritchard, O. Hassan, and K. Morgan. An efficient marker and cell solver for unstructured hybrid meshes. In *Proceedings of the 16th International Conference on Finite Elements in Flow Problems, March 23-25, Munich, Germany*, 2011.

[75] K. A. Sørensen. *A multigrid procedure for the solution of compressible fluid flows on unstructured hybrid meshes*. PhD thesis, University of Wales, Swansea, 2002.

[76] Z. Zhang. *The Simulation of 3D Unsteady Incompressible Viscous Flows with Moving Boundaries on Unstructured Meshes*. PhD thesis, Swansea University, 2007.

[77] G. D. Smith. *Numerical Solution of Partial Differential Equations: Finite difference methods*. Oxford University Press, 1999.

[78] E. Suli and D. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, 2006.

[79] B. M. Irons. A frontal solution program for finite element analysis. *International Journal for Numerical Methods in Engineering*, 2:5–32, 1970.

[80] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, 1986.

[81] 2nd. Ed., editor. *Numerical Recipies in Fortran: The art of scientific computing*. Cambridge University Press, 1992.

[82] I. S. Duff and J. K. Reid. The mulitfrontal solution of indefinite sparse symmetric linear euqations. *ACM Transactions on Mathematical Software*, 9:302–325, 1983.

[83] I. S. Duff. Ma57 - a code for the solution of sparse symmetric definite and indefinite systems. *ACM Transactions on Mathematical Software*, 30:118–144, 2004.

[84] T. A. Davis adn I. S. Duff. A combined unifrontal/mulitfrontal method for unsymmetric sparse matrices. *ACM Transactions on Mathematical Software*, 25:1–20, 1999.

[85] W. H. Liu. The multifrontal method for sparse matrix solution: Theory and practical. *SIAM Review*, 34:82–109, 1992.

[86] Hsl, a collection of fortran codes for large-scale scientific computation, see http://www.hsl.rl.ac.uk/.

[87] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[88] Y. Saad and H. A. van der Vorst. Iterative solution of linear systems in the 20th century. *Journal of Computational and Applied Mathematics*, 123:1–33, 2000.

[89] Y. Saad. *Interative Methods for Sparse Linear Sytems*. PWS Pub. Co., Boston, 1996.

[90] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. 1994.

[91] J. A. Meijrink and H. A. van der Vorst. An interative solution method for linear systems of which the coefficient matrix is a symmetric m-matrix. *Mathematics of Computation*, 31:148–162, 1977.

[92] O. Soto, R. Lohner, and F. Camelli. A linelet preconditioner for incompressible flow solvers. *International Journal of Numerical Methods for Heat & Fluid Flow*, 13:133–147, 2003.

[93] P. Chin, E. F. D'Azevedo, P. A. Forsyth, and W. P. Tang. Preconditioned conjugate gradient methods for the incompressible navier-stokes equations. *International Journal for Numerical Methods in Fluids*, 15:273–295, 1992.

[94] R. Aubry, F. Mut, R. Löhner, and J. R. Cebral. Deflated preconditioned conjugate gradient solvers for the pressure-poisson equation. *Journal of Computational Physics*, 227:10196–10208, 2008.

[95] M . Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 182:418–477, 2002.

[96] R. Lohner. *Applied CFD Techniques: An Introduction based on Finite Element Methods.* Wiley, 2001.

[97] F. Aurenhammer and H. Edelsbrunner. An optimal algorithm for constructing the weighted voronoï diagram in the plane. *Pattern Recognition*, 17:251–257, 1984.

[98] Z. Xie, O. Hassan, and K. Morgan. Tailoring unstructured meshes for use with a 3d time domain co-volume algorithm for computational electromagnectics. *International Journal of Numerical Methods in Engineering*, 2010.

[99] S. Walton, O. Hassan, K. Morgan, and M. R. Brown. Modified cuckoo search: A new gradient free optimisation algorithm. *Chaos, Solitons and Fractals*, 44:710–718, 2011. Awaiting publication.

[100] S. Walton, O. Hassan, and K. Morgan. Using proper orthogonal decomposition to reduce the order of optimization problems. In W. A. Wall and V. Gravemeier, editors, *Proceedings of the 16th International Conference on Finite Elements in Flow Problems, March 23-25, Munich, Germany*, page 90, Munich, 2011.

[101] B. Munson, D. Young, and T. Okiishi. *Fundamentals of Fluid Mechanics.* Wiley, 5th edition, 2006.

[102] B. F. Armaly, F. Durst, J. C. F. Pereira, and B. Schonung. Experimental and theorectical investigation of backward-facing step flow. *Journal of Fluid Mechanics*, 127:473–496, 1983.

[103] J. H. Lienhard. Synopsis of lift, drag and vortex frequency data for rigid circular cylinders. Technical report, Washington State University, 1966.

[104] X. Zhang, S. Ni, and G. He. A pressure correction method and its applications on an instructured chimera grid. *Computers and Fluids*, 37:993–1010, 2008.

[105] P. De Palma, M. D. de Tullio, G. Pascazio, and M. Napolitano. An immersed-boundary method for compressible viscous flows. *Computers and Fluids*, 35:693–702, 2006.

[106] R. Codina, H. Coppola-Owen, P. Nithiarasu, and C. B. Liu. Numerical comparison of cbs and sgs as stabilization techniques for the incompressible navier-stokes equations. *International Journal for Numerical Methods in Engineering*, 66:1672–1689, 2006.

[107] M. N. Linnick and H. F. Fasel. A high-order immersed interface method for simulating unsteady incompressible flows on irregular domains. *Journal of Computational Physics*, 204:157–192, 2005.

[108] C. Liu, X. Zheng, and C. H. Sung. Preconditioned multigrid methods for unsteady incompressible flows. *Journal of Computational Physics*, 139:35–57, 1998.

[109] H. Pan and M. Damodaran. Parrallel computation of viscous incompressible flows using godunov-projection method on overlapping grids. *International Journal For Numerical Methods in Fluids*, 39:441–463, 2002.

[110] C. Kiris and D. Kwak. Numerical solution of incompressible navier-stokes equations using a fractional-step approach. *Computers and Fluids*, 30:829–851, 2001.

[111] A. Uranga, P. Persson, M. Drela, and J. Peraire. Implicit large eddy simulation of transitional flows over airfoils and wings. In *AIAA 2009-4131*, 19th AIAA Computational Fluid Dynamics, San Antonio, Texas, 2009.

[112] A. Uranga, P. O. Persson, M. Drela, and J. Peraire. Implicit large eddy simulation of transition to turbulence at low reynolds number using a discontinuous galerkin method. *International Journal for Numerical Methods in Engineering*, 87:232–261, 2011.

[113] M. Zhu, Y. Shimizu, and N. Nishimoto. Calculation of curved open channel flow using physical curvilinear non-orthogonal co-ordinates. *International Journal for numerical methods in fluids*, 44:55–70, 2004.

[114] T. Lehnhäuser and M. Schäfer. Efficient discretization of pressure-correction equations on non-orthogonal grids. *International Journal for Numerical Methods in Fluids*, 42:211–231, 2003.

[115] G. S. B. Lebon, M. K. Patel, and K. A. Pericleous. Investigation of instabilities arising with non-orthogonal meshes used in cell centred elliptic finite volume computations. *Journal of Algorithms & Computational Technology*, 6:129–152, 2011.

[116] D. McBride, T. N. Cross, and M. Cross. A coupled finite volume method for the solution of flow processes on complex geometries. *International journal for numerical methods in fluids*, 53:81–104, 2007.