UNIVERSITY OF SOUTHAMPTON

Propeller Tip Vortex Simulation Using Adaptive Grid Refinement Based On Flow Feature Identification

by

Christos Pashias

Doctor of Philosophy

School of Engineering Sciences

August 2005

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

SCHOOL OF ENGINEERING SCIENCES

Doctor of Philosophy

PROPELLER TIP VORTEX SIMULATION USING ADAPTIVE GRID REFINEMENT BASED ON FLOW FEATURE IDENTIFICATION

By Christos Pashias

In this thesis a novel 2-D vortex identification scheme is refined and extended to 3-D. The developed method is applied to a 3-D wing and two marine propellers; DTMB P4119 and INSEAN E779A. In addition a wake identification scheme is developed and applied to a 3-D wing.

The vortex identification method is based on a simple mathematical scheme applied on 2-D planes. The vortex is identified by locating the point closest to the most variance in the velocity direction. The method is extended to 3-D by the use of a series of planes. Multiple vortices can be identified using this method.

The vortex and wake identification schemes have been applied in conjunction with adaptive grid refinement on a 3-D wing showing improved agreement with experimental results. The vortex dependency on grid size has been demonstrated.

The vortex identification scheme has been extended such that it can identify complex vortex core lines, typical of marine propellers. The method has been successfully applied to two marine propellers and the results compared to experiments. An improved agreement has been demonstrated for the grid adapted cases.

Table of Contents

List of Figures

List of Tables

Nomenclature

ρ	kgm^{-3}	Density
ϕ	-	Pitch angle
ψ	-	Helix angle
μ_{xi}, μ_{yi}	m	co-ordinates of the cluster centre
P	m	Propeller pitch
p	Pa	Pressure
p_0	Pa	Reference pressure
r	m	Local radius
V_R	ms^{-1}	$=\sqrt{V^2+(2\pi nr)^2}$
C_P	-	$=\dfrac{(p-p_0)}{\frac{1}{2}\rho V_R^2}=1-\left(\dfrac{V}{V_R}\right)$
b	m	Span
c	m	Chord
V_a	ms^{-1}	Advance speed
V_x	ms^{-1}	Axial velocity
V_t	ms^{-1}	Tangential velocity
V_r	ms^{-1}	Radial velocity
L	N	Lift
D	N; m	Drag; Propeller diameter
Δn	m	Distance between 1st and 2nd grid point
C_L	-	$=\dfrac{L}{\frac{1}{2}\rho S V^2}$
C_D	-	$=\dfrac{D}{\frac{1}{2}\rho S V^2}$
K_T	-	$=\dfrac{T}{\rho n^2 D^4}$
K_Q	-	$=\dfrac{Q}{\rho n^2 D^5}$
J	-	$=\dfrac{V_a}{nD}$
l_n	m^{2n}	$l_n=\dfrac{\sum_0^{n-1} r_n^2}{n_{\beta\_exist}-1}$
n	s^{-1}	Revolutions per second
N_c	-	Number of clusters
N_{pi}	-	Points in ith cluster
Q	Nm	Torque
q_θ	ms^{-1}	Tangential velocity
S	m^2	Planform area
T	N	Thrust
τ_ω	Pa	Wall shear-stress
U_∞	ms^{-1}	Free stream velocity
x_s, y_s	m	2-D section co-ordinates
y^+	-	$y^+=\dfrac{\sqrt{\tau_\omega/\rho}.\Delta n}{\nu}$

Acknowledgements

Stephen R Turnock, my supervisor, for his guidance, patience and countless hours he has invested in this research. Without him this research would not have been possible.

AF Molland, for his advice, based on his invaluable experience that has helped on numerous occasions.

Overseas Research Scholarships Scheme, for partly funding my research.

My fellow researchers, for sharing their experiences and advice when going through the same problems they encountered in the past.

Emma M Barnett, for being there for me during the numerous obstacles encountered during the period of the research and for nagging me to complete this work.

Stephen and **Anne Barnett**, for providing me with a home away from home.

Charis and **Emilia Pashias**, my parents, for giving me the opportunity to study and their financial support throughout my University years.

1 Introduction

1.1 Aims and objectives

The work done on a fluid by a marine propeller generates a rotating propeller wake where the shed vorticity rapidly coalesces into a tip vortex and a hub vortex system for each blade. The accuracy of the computational prediction of propeller performance or indeed any lifting surface is influenced strongly by the accuracy with which this process of vortex formation is captured.

Current numerical methods are capable of capturing such flows but require a very fine mesh in the appropriate regions of the flowfield. Efficient meshes, where an appropriate mesh density is chosen for the local flowfield, cannot be generated *a priori*. Adaptive meshes offer a solution to this problem but identifying the vortex is a challenging task, with current methods being either complex with problems of robustness or unable to identify the correct regions for refinement.

The aim of this research was to develop an adaptive refinement method suitable for the detailed modelling of vortical flows capable of being applied for the prediction of propeller performance. The method must be capable of identifying multiple vortices within the flow with minimal computational effort and without any user interaction. The specific objectives are:

- to investigate methods of applying the existing two dimensional VORTFIND method [1, 2] for predicting vortex cores to the complex three dimensional flows found at the tip of a lifting surface and in particular the predominantly helical flow field associated with marine propellers;
- to investigate adaptive refinement schemes suitable for use with the above method;
- to apply the developed method to a control surface to capture the tip vortex flow in order to test, validate and refine the method before applying it to the complex flow field of marine propellers; and
- to apply the method to two standard marine propellers which have been extensively tested and used widely for numerical method validation.

The methods developed in this research will provide an improved computational tool that can be applied for the design and optimisation of the next generation of marine propellers.

1.2 Introduction

1.2.1 A brief history of the marine propeller

Screws can be dated back to ancient times. Archytas, a Greek mathematician, student of Pythagoras and friend of Plato, is credited for inventing the screw when he put an inclined plane on a cylinder about 400 B.C. In 220 B.C. Archimedes famously used a screw to lift water. The first to suggest the use of a screw for propulsion was Leonardo Da Vinci in 1480-1510 when he sketches a helicopter. In 1752 Bernoulli is the first to suggest propelling of boats using *"vanes set at an angle of 60^o to both the arbor and the keel"*. The first to use a screw to propel a marine vessel was David Bushnell to drive his submarine *Turtle* in 1776. From then on there are many noteworthy applications of simple screws.

However, it was not until the 1800s, with the advent of the steam engine, when propellers started to replace sail power for commercial shipping. Since then the propeller has dominated the propulsion for marine transportation, and still does so today, with almost all commercial ships propelled by screws.

Their design has changed over the centuries from Archimedes's wooden screw to modern high performance composite propellers [3]. They are one of the most widely used devices for producing thrust in a fluid medium; still used for aeroplanes, helicopters and ships just to name a few. Sizes range in diameter from a few centimetres used on models, to the world's largest propeller at 9.1 metres for a large container ship.

However, as the requirements imposed on propellers have changed over the years new research is required to meet those needs. The power transferred through propellers has increased in recent years and vessels can now achieve higher speeds with a resultant risk of cavitation. These more heavily loaded propellers create stronger tip vortices, which must be modelled accurately in order to predict the performance of and design these modern propellers. The dynamic behaviour of the tip vortex can be responsible for vibration and noise which is becoming increasingly important in passenger vessels [4]. The trend of modern propeller design toward high blade tips has succeeded in reducing the pressure amplitudes in general, but is often considered to cause a rise in pressure pulses of higher order. An explanation can be found in the occurrence of less, but fluctuating, strong and bursting tip vortex cavitation [5]. Thus is it very important to improve tip vortex modelling [6] and develop the tools necessary to design the next generation of propellers.

1.2.2 Background theory

The basic principles of propeller operation are well understood [7, 8]; however the detailed flow and physics, necessary to fully comprehend propellers, are still in constant development and highly complex. The challenges of accurately modelling propellers still fascinate and perplex enough to justify the resources of many researchers today. In particular, marine propellers present more of a challenge than their aerospace counterparts. Even though they both operate in fluids with similar governing physics at low speeds the marine propeller still has many aspects that complicate matters. In most cases, airscrews operate in uniform flow ahead of any obstructions, whereas marine propellers operate in the wake of hulls requiring unsteady simulations to capture their behaviour correctly. In addition to their vicinity to the free surface cavitation issues can arise. Modelling cavitation, on its own provides a challenge for current methods [4, 6]. Also the restriction in diameter combined with cavitation considerations for blade areas leads to much lower aspect ratio blades than their aerospace equivalents. The heavy loading combined with the low aspect ratio results in significantly increased importance in tip vortex modelling.

The original theory, as first formulated by Rankine [9] excluded the viscous effects, the rotation of the slipstream, and the uneven load distribution, with the scope of evaluating the ideal efficiency of such a propulsive system (also called *actuator disc*). The rotor is degenerated into a disc perpendicular to the thrust, and is capable of sustaining a pressure difference between its two sides, and imparting linear momentum to the fluid that passes through it. The mechanism of thrust generation requires the evaluation of the mass flow through a stream tube bounded by the disc.

In 1878 William Froude developed the theory of a propeller blade's elements, which reflects the generated efforts on each section of the blade [10]. However it was not until Betz's [11] work in 1919 and later Goldstein's [12] in 1929 employing Prandtl's [13] lifting line theory that showed optimum propellers could be designed. Prandtl assumed that the 3-D problem could be solved by concentrating the circulation around the blades on individual lifting lines and that the flow on each line can be regarded as 2-D. Using Goldstein's solution for the optimum propeller in uniform flow with 2-D experimental section data optimum aircraft propellers could be designed. This approach is successful for high aspect ratio blades for which the underlying assumption that the flow is principally 2-D is more or less valid.

However for the low aspect ratio blades widely used for marine propellers this assumption is not valid. It was not until 1952, when Lerbs [14] published his paper on the extension of Goldstein's lifting line theory for propellers with arbitrary radial distributions of circulation in both uniform and radially varying inflow, when at last marine propellers could be modelled with some degree of accuracy. Although its acceptance was slow, it still is, even today, universally accepted as a good procedure for establishing the principal characteristics of the propeller at an early design stage.

The next major improvement in propeller modelling was the use of numerical lifting-surface methods. Now the skew and the radial distribution of circulation could be modelled. The formulation was published by Sparenberg in 1959 [15]. This led to a burst in publications in 1961 and 1962 of computer based propeller lifting surface codes. Most notable examples were Pien [16], Kerwin [17], van Manen & Bakker [18] and English [19]. However due to the limited computing power of that era these methods incorporated simplifying assumptions which since have been found unnecessary with the rapid development of high performance computers. The basic formulation however is essentially the same, Brockett 1981 [20] and Greeley & Kerwin 1982 [21].

The above methods, although suitable for design purposes, provided limited information on the section flow. Boundary element methods (BEM) or panel codes as they are popularly known today, can model more realistic geometries taking into account the section shape and thickness. In 1985 Hess & Valarezo [22] developed a BEM for propellers based on Hess's [23] lifting method. Since then the popularity of panel codes has been widespread in all aerodynamic and hydrodynamic fields. The computational resources available today are such that detailed optimisation studies can be carried out within reasonable timescales even on powerful personal computers [24]. However there are many underlying simplifications in their formulation and as always, with the relentless development in computing power things will move on quickly to more complex methods such as Reynolds Averaged Navier-Stokes (RANS) codes for design purposes.

Most work on RANS methods was done in the late 1980's and improved greatly in the early 1990's. The initial work was undertaken by Kim & Stern [25] in the late

80's and they showed the potential of such methods, although for an unrealistic propeller geometry due to the limited computing resources available at the time. In the 1990's researchers such as Uto [26] and Stanier [27] developed methods for realistic propeller geometries with detailed flow features. Unsteady viscous computations were first undertaken by Chen et al. [28] and showed the applicability of such methods for unsteady propeller flows albeit obtaining poor results due to the limited mesh size feasible at the time. Maksoud *et al.* [29] performed unsteady calculations for a propeller operating in the wake of a ship using a non matching multi-block scheme.

As indicated by Stanier [30], Bull [31] and Maksoud *et al.* [32] and as will be shown later in this work mesh quality and local density are key factors in obtaining good results with RANS codes. In particular, it is essential to have a good mesh topology in the vicinity of the tip vortex region as this has a strong influence on the developed thrust and required torque.

1.3 Vortices

Vortices are present in most fluid flows and their behaviour can be considered to be of fundamental importance A vortex is the rotation of multiple particles around a common centre. They should not to be confused with vorticity which is used as a measure of the rotationality present at a location within the flow.

Vortices come in all shapes and sizes: large scale vortices are responsible for tornadoes and the behaviour of galaxies; medium scale vortices affect the characteristics of most engineering structures, such as aircraft, ships and buildings; small scale vortices are the building blocks of turbulent flow.

1.3.1 Vortices and nature

Vortices in nature have affected the evolution of many animals. Birds have fingered feather wing tips to eliminate the effect of the tip vortex. Dolphins and other fast fish have scimitar shaped fins as a solution to the same problem. Dolphins go one step further by controlling the development of the turbulent boundary layer to reduce friction drag. The resilient dolphin shin acts like a damper to absorb oscillatory energy from the boundary layer and delay transition [33].

If a force is powerful enough to shape nature it will influence engineering design as well. As always man tries to copy nature in his inventions. Elliptical planform wings were designed to reduce induced drag caused by the tip vortex. The

Spitfire was superior to its opponents partly for this reason [34]. Then winglets appeared on aircraft wing tips and yacht keels again influencing the strength and location of the tip vortex system [33]. In addition, an artificial dolphin skin (Laminflo [35]) has been tested on torpedoes giving a large increase in laminar flow with a substantial decrease in the friction coefficient from 0.0026 to 0.0011[33]. Just as vortices have shaped nature over millions of years, vortices are today shaping man made structures. As the human knowledge and understanding of vortices evolves so do the shapes designed by engineers.

1.3.2 Structure of a plane vortex

There are two basic types of plane vortices: one where the velocity is slower at the centre than the sides and another where it is faster at the centre. This results in two basic types of velocity distribution as described below:

1. Consider a solid disc that rotates steadily around an axis through its centre. The velocity of points on the disk increases linearly with distance from the centre (Figure 1.1). Imagine the disc is hollow and it is filled with fluid. If the experiment is repeated, the fluid will also rotate like a solid body, apart from a transition period at the start. This is due viscous effects. Hence the velocity of the fluid particles increases linearly with distance from the axis of rotation. This fluid motion is called 'solid-body rotation' [36].
2. Consider a long circular rod rotating in a fluid. The highest velocity in the fluid will occur on the surface of the rod where the fluid has the same velocity as the rod due to viscous effects. Away from the surface of the rod the velocity diminishes in inverse proportion to the radial distance squared (Figure 1.2). The centrifugal force pushing outwards due do the rotation is balanced by the force due to the pressure gradient. This is called a 'potential vortex' [36] as it is irrotational and can be expressed in terms of a scalar potential.

An important difference between solid-body rotation and potential vortices is that a potential vortex has no vorticity whereas solid-body rotation has constant angular velocity and vorticity. In the case of the potential vortex the rod will still experience a sold-body rotation. If the rod is replaced by fluid that is also experiencing solid-body rotation a Rankine vortex is created (Figure 1.3). A Rankine vortex is a good representation of a real vortex. A real vortex has a region where the velocity varies linearly with distance from the centre. This is due to viscosity, and the region is called the viscous core. Outside the viscous core the velocity varies inversely proportional with distance squared. Capturing this regime transition with a RANS code is difficult due to the fine mesh resolution required to capture the core region which is dominated by viscous effects just like a boundary layer. In addition, enough of the vortex decay must be captured correctly in the potential flow region since this influences the overall vortex structure.

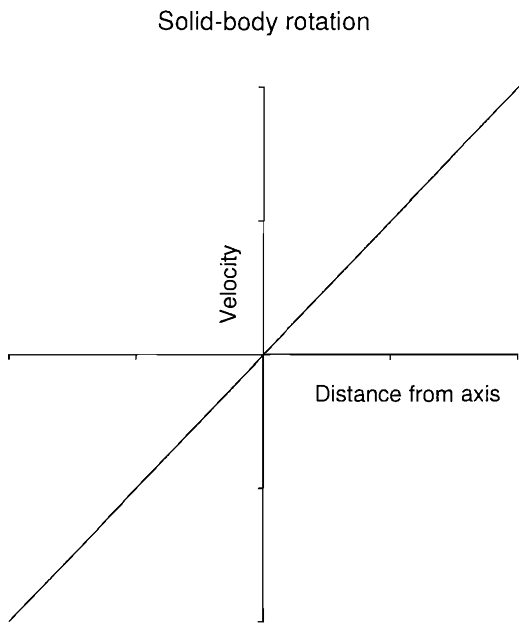


Figure 1.1 – Solid-body rotation. Velocity varies linearly with distance

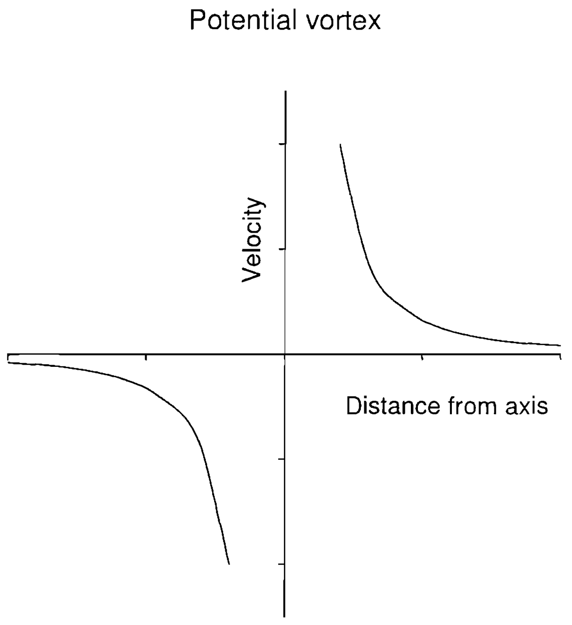
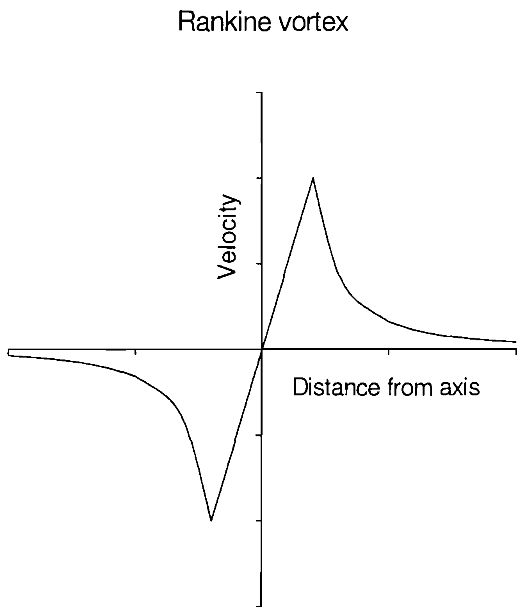


Figure 1.2 – Potential vortex. Velocity varies inversely proportional to distance squared



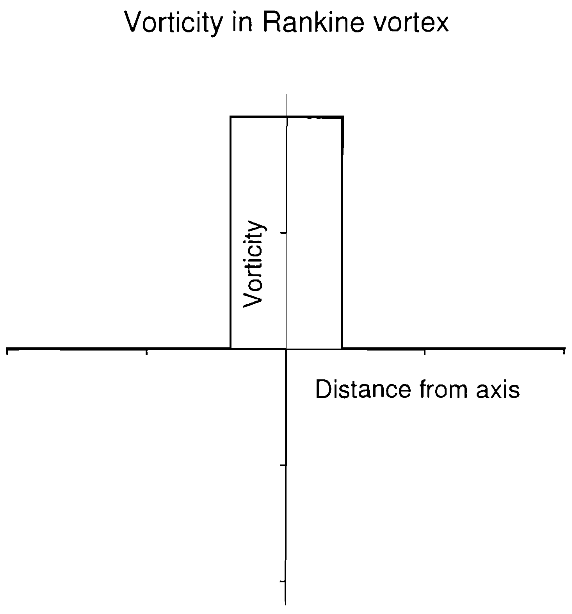


Figure 1.3 – Rankine vortex. Consists of solid-body rotation and potential vortex. Vorticity exists only in the solid-body rotation region.

1.3.3 Tip vortices

A foil experiencing three-dimensional flow has very different characteristics than a foil experiencing two-dimensional flow. The flow will tend to spill over the free ends from the positive pressure side to the negative pressure side. Such a flow removes the pressure difference at the tips of the foil and decreases it over the entire span.

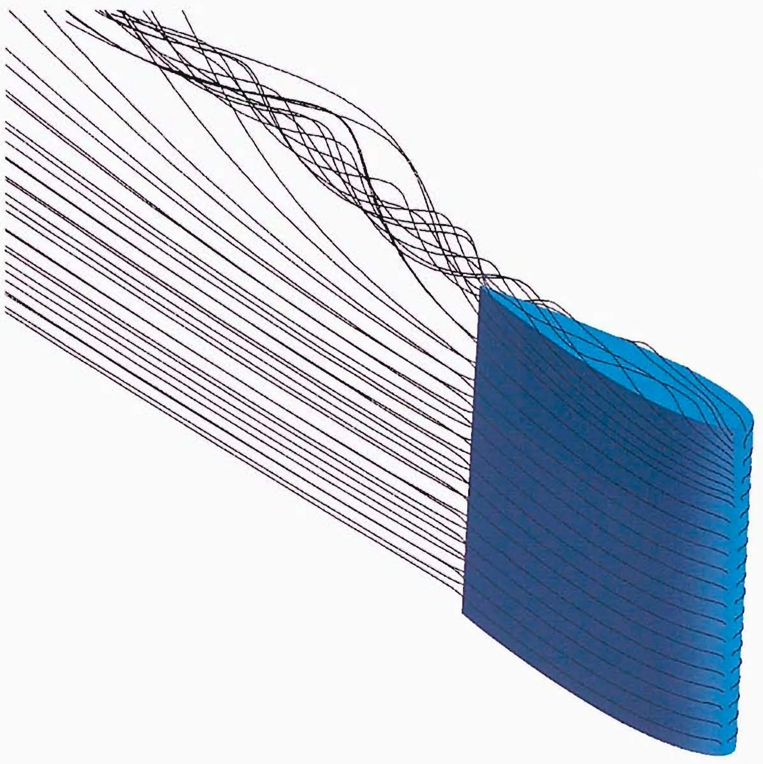


Figure 1.4 – Tip vortex of 3D wing

The tip vortex also influences the flow over the entire lifting surface. Near the tip the surface pressure is influenced by the reduction in pressure within the tip vortex, especially on the suction side of the foil where the vortex is located. In addition a downwash is created behind the trailing edge of the foil due to the vortex. Aerodynamicists try to eliminate these effects or sometimes even exploit them, as is the case with delta wing planforms operating at high angle of attacks. In this case the leading-edge separation, which induces a non-linear lift increment called vortex-induced lift, is highly dominant on delta wings [36].

In 1897 Lanchester realised that the tip vortex has a detrimental effect on wings and secured a patent covering the use of end plates. In his book *Aerodynamics*

[37] he describes the structure of the tip vortex. Since then there have been many studies of tip vortex flows and its influence on finite foils. Prandtl [38, 39] modelled the effect of the tip vortex on the flow of a finite foil published in 1918 using his lifting line theory.

Many methods to reduce the detrimental effect of the tip vortex have been devised, from simple elliptical planforms to more complex tip winglets such as on aircraft and yacht keels. In order to design and develop these tip vortex control devices, numerical models of the flow near the tip of the foil are required

1.3.4 Vortex capturing using a fixed (structured) mesh

In order to capture vortical flow a numerical model capable of solving the flow characteristics at any point within it is required. Such numerical methods are based on solutions to the Euler (with zero viscosity) or the Reynolds Averaged Navier-Stokes (RANS) codes based on some form of turbulent closure. Euler and RANS have until recent years been limited to small simple cases due to limited computing power. Recent advances in computational power have led to several studies capturing numerically predicted vortex flows [40]. A large amount of development has been done resulting in progressively more sophisticated and robust models.

A sufficiently fine grid spacing is required in the region of the vortex core [41] to capture the vortex. Dacles-Mariani and Zilliac [41] used a structured grid to capture the tip vortex of a foil. A grid dependency study was carried out using an analytical vortex. The results showed that at least 15-20 grid points in the viscous core region are needed to capture the vortex correctly. In addition the vortex is more sensitive to cross flow plane grid refinement than streamwise refinement. This means that the cells can be stretched in the streamwise direction in an attempt to keep the grid size down. An investigation of the influence of turbulence model indicated the difficulties in capturing the vortex. Good agreement with measured results was found using 1.5 million grid points. The velocities were within 3% but the vortex core static pressure was under predicted. This under prediction was accounted for by limitations coming from the turbulence modelling. This is in agreement with later studies carried out by Viot *et al.* [42]. This suggests that the turbulence model is critical to the results and that a more suitable model should give better results.

Spall used a structured grid to solve the Euler equations for a NACA0012 rectangular wing [43]. A multi-block grid was used in an attempt to cluster the grid points near the vortex core. The relative small vortex core diameter of 0.04c, proved a challenge to cluster the grid in the correct region. The results showed that a slight displacement of the vortex core outside this region results in a considerable degradation in the solution. Results are presented for 1.5 million grid points with 10-18 across the vortex core. The vortex core radius was dependent on grid spacing, halving from a grid using 372,016 cells to 1,986,000 cells.

Berntsen *et al.* [44] used an Euler code with a structured multi-block grid to model tip vortex cavitation. First a crude calculation was performed to derive the general shape of the vortex. Following this investigation, the grid was clustered near the centre of the vortex. This was repeated three times resulting in a grid in excess of 0.5 million cells. The resulting grid is good but requires a large amount of effort and skill to generate. The process cannot be easily automated even for standard cases. Visually the results were in good agreement with respect to cavity length. However vortex core pressures were not predicted accurately.

Hsiao and Chahine [45] used a 12-block structured mesh to model a NACA16020 finite-span elliptic hydrofoil including a dynamics bubble model for cavitation. The mesh used consisted of 2.7 million cells and the mesh was regenerated after each solution with modified clustering to ensure that 16 cells were always present across the vortex core. An unspecified number of mesh regeneration steps were required to cluster the grid in the vortex region.

From the above studies it is clear that to capture the vortex flow accurately a fine grid spacing in the region of the vortex core is required. The reason is that in the vortex core the pressure decreases rapidly. The vortex core pressure depends on the vortex core's radius [33]. A small change in this radius results in a big change in the minimum pressure. To capture the vortex radius correctly a fine grid spacing is required in the vicinity of the vortex core. The computational resources required to generate a uniform grid with fine enough grid spacing are impractical. Using a coarser grid with clustering of the fine mesh near the vortex improves the results without a large increase in cost. The problem with such methods is that the location of the vortex must be known to generate the mesh. This either requires solving the problem in advance or a very detailed understanding of the flow to predict *a priory* the vortex position. With either route, the vortex is not guaranteed to lie in the predicted area and the mesh refinement will change the behaviour of the vortex and thus its position. These effects are compounded as the vortex evolves downstream.

Vorticity confinement is a method to conserve and concentrate vorticity on a regular grid [93]. It prevents the dissipation of vortical structures on coarse grids. This method has been used by Löhner *et al.* to track vortices over large distances [94]. The results show that the vortex is maintained further downstream using the vorticity confinement method on coarse grids. However many researchers have expressed concerns over the validity of this method since the vorticity confinement term acts as a body force altering the conservation of momentum.

Although the above methods work, the grid cannot be generated *a priori*. The solution to this problem is to use an adaptive scheme to refine the grid during the solution process. Adaptive grids have been used successfully in many different flows, to capture flow characteristics such as shockwaves in transonic flight [46, 47].

1.3.5 Adaptive Meshes

For well behaved problems a grid of uniform mesh spacing gives satisfactory results. However, there are classes of problems where the solution is more difficult to estimate in some regions (perhaps due to discontinuities, steep gradients, shocks, etc.) than in others. A uniform grid can be used which has a spacing fine enough such that the local errors estimated in these difficult regions are considered to be acceptable. But this approach is computationally costly especially in three dimensions. In addition, for time dependent problems it is difficult to predict in advance a mesh spacing that will give acceptable results. The goal of mesh adaptation is the determination of the optimum mesh-point distribution that results in equipartition of the error for each individual simulation thus giving an optimum solution for a specified grid size.

In adaptive grids the idea is to have grid points moved/inserted as the physical solution develops, concentrating in regions of large variation in the solution as they emerge. A base coarse grid is used as the starting point. As the solution proceeds the regions requiring more resolution are identified by some parameter characterizing the solution. A finer subgrid is superimposed only in these regions. Finer and finer subgrids are added recursively until either a given maximum level of refinement is reached or the local truncation error has dropped below the desired level.

There are many methods of mesh refinement [46, 48, 49]. One way is to regenerate the mesh after every refinement cycle with grid points clustered in the areas identified by the mesh criterion. This is time consuming and the solution might have to be initialised again.

The second method is to subdivide the cells in the areas where mesh refinement is required into smaller cells, thus decreasing the grid spacing. However the grid quality is dependant on the initial mesh, unless a smoothing algorithm is applied [50]. The other advantage is that the solution can be linearly interpolated at the new grid points for the next iteration, thus speeding up the solution. In addition there is no need to regenerate the entire mesh.

Apart from the methodology used to generate the different refined levels of the grid, the other important factor when using adaptive refinement techniques is the criterion used to adapt to. The mesh can be refined using any variable or combinations of variables. Choosing a criterion suitable for the flow problem and what is trying to be achieved is important since it will influence the final solution [51]. One of the most frequently used schemes is an error based criterion approach.

Löhner [51] describes the components of such an adaptive refinement scheme. Different error indicators/estimators are discussed as well as different mesh refinement techniques for unstructured meshes. Results are presented for several test cases. Nithiaratsu *et al.* [48] present the results for a transonic aerofoil and lid driven cavity using different error indicators. From the presented results is it concluded that for most cases a gradient based error indicator yields better results than a curvature based indicator. Pelletier [52] also uses an error based adaptation criterion to obtain a grid independent solution. He describes the required qualities of an error estimator. Results are presented for a 2D aerofoil and a backward facing step.

Adaptive refinement can be applied to both structured and unstructured grids. Structured grids have implicit connectivity, which for most cases reduces the memory requirements since no grid connectivity information needs to be stored, whereas an unstructured mesh explicitly defines the connectivity between all mesh elements. However this proves to be a major obstacle when adaptive refinement techniques are used with structured meshes. It is difficult to preserve the implicit connectivity after grid refinement. A solution to this problem is to start with a structured mesh but have explicit connectivity. This maintains some of the advantages of structures meshes and at the same time simplifies the refinement scheme. However there is another problem that has to be addressed. If each face of a cell volume in the grid must be connected to only to one other face then if a hexahedron is refined into smaller hexahedra they cannot be adjacent to an unrefined one. This can be solved in two ways: refining the hexahedra into tetrahedra, resulting in a hybrid grid or allow a face to be connected to multiple faces and perform what is termed as hanging node adaptation. If hanging node meshes are not an option then the split through a hexahedral cell due to mesh refinement has to propagate throughout the structured mesh to maintain the connectivity of the mesh. This results in refined cells in areas other than the ones of interest and unnecessarily large mesh sizes.

Unstructured grids are ideal for mesh refinement techniques since the connectivity of the grid is explicit [46, 49]. Cells can be subdivided into numerous new cells without having to worry about the connectivity structure of the mesh.

When meshing complex geometries it is easier to use unstructured grids. Unstructured grids can be generated automatically from a set of surfaces using various schemes [53]. It is possible to generate structured grids for most problems but a complicated multi-block structure has to be used for most cases. Even for a seemingly simple elliptic foil, 16 blocks had to be used in order to model the tip correctly [44]. Apart from the treatment of complex geometries, the second advantage of unstructured meshes is the ease with which solution-adaptive meshing may be implemented. Since no inherent structure is assumed in the representation of the mesh, mesh points may be added, deleted, or displaced, and the mesh connectivity may be locally reconfigured in the affected regions.

One possible way to overcome the limitations and problems imposed by structured grids is to use an overset method. Grids of different densities are laid over each other with no connectivity restrictions. This enables the used of structured meshes for complex geometries such as an aircraft in landing configuration as performed by Rogers *et al* [54]. A good reference of the capabilities and past applications for overset techniques is given by Chan *et al* [55]. Unstructured overset grids are possible but such schemes are still in development [54].

For tip vortex capture it is possible to overlay a finer grid about the identified vortex core in order to capture the vortex. If such an approach is used structured O-grid would be the most suitable topology. However Chan *et al* [55] state that overset methods require substantial user interaction and are laborious for complex geometries.

1.3.6 Vortex Identification Methods

For the successful application of an adaptive scheme, a suitable variable or variables must be selected for use as the adaptation criterion/criteria. Pressure gradients and local error functions have been used successfully. Such techniques are suitable for most flows but fail to resolve flow features such as vortices adequately for a given grid size [56]. In order to identify vortices several methods have been developed and applied to a multitude of problems. The ten methods given below are by no means a complete listing of vortex identification algorithms but are considered to represent the current state of the art.

- *Helicity* Method by Levy [57]
- *Vorticity Maxima* Method by Strawn et al. [58]
- *Streamline* Method by Sadarjoen et al. [59]
- *Swirl Parameter* Method by Berdahl and Thompson [60]
- λ_2 Method by Jeong and Hussain [61]
- *Predictor-Corrector* Method by Banks and Singer [62]
- *Eigenvector* Method by Sujudi and Haimes [63]
- *Parallel Vectors* Method by Roth and Peikert [64]
- *Combinatorial* Method by Jiang et al. [65]
- *Vortfind* Method by Pemberon [1, 2]

Almost every published work carried out on vortex identification presents a classification of the methods developed by its predecessors. Here the methods are classified using three taxonomies as presented by Jiang *et al.* [66]

Method	***Region\Line***	***Galilean***	***Local\Global***
Helicity	Line	Not invariant	Local
Vorticity Maxima	Line	Invariant	Local
Streamline	Region	Not invariant	Global
Swirl Parameter	Region	Not invariant	Local
λ_2	Region	Invariant	Local
Predictor-Corrector	Line	Invariant	Global
Eigenvector	Line	Not Invariant	Local
Parallel Vectors	Line	Not Invariant	Local
Combinatorial	Region	Not Invariant	Local
Vortfind	Line	Not Invariant	Global

Table 1-1 - Taxonomy of Vortex detection algorithms

The first taxonomy classifies detection methods based on the definition of the identified vortex. A vortex can be defined either as a region or as a line. A region-based vortex definition specifies a group of cells that lie in the vortex region. A line-based vortex definition, on the other hand, is a set of lines describing the vortex core line. In general, region-based algorithms are easier to implement and computationally cheaper than their line-based counterparts. Line-based algorithms must precisely locate points to describe the vortex core line. However, line-based algorithms provide more compact representations of vortices and can easily distinguish between multiple vortices. The latter is problematic for region-based approaches [66].

The second taxonomy classifies detection methods based on whether or not they are Galilean (Lagrangian) invariant. In a time varying flow field, a vortex exhibits swirling motion only when viewed from a reference frame that moves with the vortex [36]. A detection method is Galilean invariant if it produces the same results when a uniform velocity is added to the existing velocity field.

The third taxonomy classifies detection methods based on whether the identification process is of local or global nature. A detection method is considered to be local if the identification process requires only operations within the local neighborhood of a grid cell. On the other hand, a global method requires examining many grid cells in order to identify vortices.

Each of the above methods has its advantages and disadvantages. Pressure minimum methods find elongated regions of low pressure which usually indicate a vortex core [62]. However minimum pressure does not always coincide with the vortex core. In addition, regions of low pressure also exist in other features of many flows which complicate the process further. Isosurfaces of low pressure are very effective when capturing a single vortex in unobstructed flows. However when multiple vortices exist the pressure surfaces become indistinct.

Methods using the eigenvalues of the velocity gradient to identify the vortex are successful; however such methods also capture many smaller structures [62]. It is common for vortex identification methods to use a combination of two criteria so as to reduce the likelihood of misclassification, which is a common problem [62].

Sujudi and Haimes [63] present a popular method based on a velocity gradient method and carry out computations on a cell by cell basis which facilitates parallel processing. Godo *et al.* [67] and Roth and Peikert [64] say that such methods fail when the vortex core line is curved such as that in turbomachinery flows. In addition methods based on vorticity magnitudes, helicity, pressure [62] or λ_2 [61]; are also dismissed by the authors.

A noteworthy method is presented by Jiang *et al.* [65] which uses a similar approach to the method developed initially by Pemberton [1, 2] and which is further developed in this work. Both methods are based on a simple analysis of velocity field on 2-D planes. They assign different values for different velocity directions and then locate the vortex by stating that a vortex is near a region with varying sector values. The difference is that Pemberton *et al.* use the distance of the different sector values to calculate a function indicating the 'distance' from the vortex whereas Jiang *et al.* flag the computational cell if it is surrounded by at least a specified number of sectors. Both methods have their advantages and disadvantages. The combinatorial method only uses cells adjacent to the reference cell or in close proximity whereas the Vortfind method is a global method. This means that the combinatorial can have reduced calculation times but also makes it prone to small localised flows which are not vortex structures as reported in Jiang *et al.* [68]. Another major drawback of the combinatorial method is that the mesh connectivity must be known. This restricts its applicability to numerical simulations since to apply the method to experimental data or data points not conforming to the grid the connectivity must be calculated retrospectively which is a computationally intensive process. This is a similar issue the gradient based methods encounter. Vortfind does not require any connectivity information and can easily be applied to experimental data or data points not conforming to grid locations. In addition the Vortfind method provides a continuous smooth function whereas the combinatorial method does not.

When the current vortex identification schemes are applied to three-dimensional flow problems they do not produce a continuous vortex core line [69, 70, 71]. This problem is over come by refining neighbouring cells as well. Assuming that the discontinuities are small this produces a more or less uniform mesh after refinement.

1.3.7 Vortex capturing using structured adaptive grids

Because of the difficulties of using adaptive schemes with structured grids such studies are limited. Hentschel [72] uses a structured C-grid around a delta wing to capture the tip vortex with a Baldwin-Lomax turbulence model. The grid refinement scheme uses grids with three different levels of refinement. Vorticity content is used as the refinement index and in the areas of interest the grid is swapped between levels. Data is interpolated between the two levels. This effectively results in a hanging node scheme. Since the finer level grids must be generated for parts of the domain that might not be required to be refined, an unnecessary memory overhead exists. In an attempt to reduce the overhead Hentschel [72] only creates finer grids for selected parts of the domain only. Although this method works, it is very difficult to apply even for simple problems.

1.3.8 Vortex capturing using unstructured adaptive grids

Most studies incorporating an unstructured grid use an adaptive grid refinement scheme. Viol *et al.* [42] used two commercial RANS codes with an unstructured grid to solve the tip vortex flow over an elliptic foil. Pressure was used to refine the coarse mesh from 280,000 to 350,000 cells. All cells having a pressure drop of more than 10% of the maximum pressure drop were refined. This resulted in a refinement area approximately twice the viscous core radius. Different turbulence models were investigated and their performance compared with experimental data. The following conclusions were made by the authors:

- The agreement between numerical and experimental results is good for all RANS codes and turbulence models tried.

- The local vortex core radii are over-estimated and thus the pressure on the vortex axis under-predicted.
- Grid refinement has a relative low influence on the vortex and reduces the vortex radii slightly.
- The κ–ε RNG and non-linear κ–ε turbulence model decreased the vortex core radii compared with the κ–ε model.

Murayama *et al.* [69, 73] used an unstructured hybrid mesh to simulate the vortex breakdown over a delta wing at high incidence. The mesh consisted of prisms in the viscous regions and tetrahedra on the rest of the domain. Pyramids had to be used at the junctions. A vortex identification technique based on critical point analysis was used to define a vortex core line. The method identifies where the velocity becomes zero [70]. The cells transversed by the vortex line were marked for refinement as well as their neighbouring cells. The vortex core line generated by this method is not continuous especially for the coarser grids. However, since the neighbouring cells are also refined this results in a continuous refined region. The refined grid consisted in excess of 0.5 million grid points and showed improvement over the original coarse grid. A comparison with calculations using second derivatives of the total pressure as the refinement criterion was made. The pressure adaptation required in excess of 0.8 million grid points to capture the vortex breakdown correctly. This was due to the fact that a larger area around the vortex was identified for refinement than using the vortex identification scheme. By selecting a different threshold value for the refinement it is possible to reduce the refined area using a pressure based approach, but choosing the correct threshold value requires trial and error. Murayama *et al.* [74, 75] applied the same method to a NACA0012 foil with similar conclusions.

1.3.9 Propeller tip vortex

Propellers are the most common form of propulsion in a fluid medium. They efficiently convert rotational energy from the engine into forward thrust. Most propellers have free blade tips. These tips create vortices which affect the thrust and torque characteristics of the propeller. The design of the blade tip will be governed by its resultant tip vortex structure. Marine propellers are restricted in diameter due to draught restrictions compared to their aircraft counterparts. This constrains the aspect ratio of the blades, which puts more of an emphasis on the induced drag. This creates a strong tip vortex which can have a detrimental effect on the performance of the propeller.

In addition, tip vortex cavitation is of major concern for marine propellers since it is an important source of noise. Cavitation can result in erosion of the propeller or even rudders placed downstream. Therefore, the track of the vortex is just as important as its strength. Recent developments with twisted rudders [76] emphasise the importance of correctly predicting the interaction between propeller and rudder.

The tip vortices generated by each blade of the propeller have a complex structure. They form helixes that vary in pitch and contract with the wake downstream of the propeller. Recent experimental work using advanced flow visualization and non-intrusive measurement techniques [77] have revealed detailed features of the vortex flow around marine propellers. However, due to the limitations of the experimental techniques the pressure field remains unknown which is crucial to the prediction of cavitation [78].

RANS computations can provide a detailed pressure field, and have been used recently to predict tip vortex flow in turbomachinery [78, 79]. Unlike tip-clearance flow, the tip vortex generated by a marine propeller is more concentrated and has a tighter structure [78], which requires a more refined grid. Only a few studies present propeller type tip vortex flow and although the thrust and torque coefficients agree well with experiments they fail to predict the vortex strength [80]. The results from [80] regardless of the turbulence model used were unlikely to have captured the tip vortex since the grid resolution was insufficient at 200,000 grid points.

A later study [78] using 2.4 million grid points showed better results. A helical domain was used with a structured mesh. The domain was helical in an attempt to cluster the grid near the estimated tip vortex position. A study with a similar approach applied to the Euler equation for a foil [43], showed that a slight displacement of the vortex core outside the cluster region resulted in a considerable degradation of the results. An adaptive procedure that can track the vortex has the potential to eliminate this problem if successfully implemented. Such a method will offer a significant improvement over non adaptive methods [56].

Dindar *et al.* [56] performed calculations for rotors using adaptive mesh techniques. They used an unstructured grid to model one blade of the rotor arrangement. Firstly an error indicator was used to refine the mesh and secondly a vortex identification technique to refine the tip vortex. Two error based refinement iterations were performed followed by two vortex identification ones. The vortex was only identified after adequate mesh refinement was carried out first with the error base scheme. They also indicated a potential problem of mesh refinement combined with parallel computations. The mesh refinement might be carried out in a region which is assigned to one processor only which will exceed the memory capacity of that single processor. In addition the error based refinement although successful, is identified as computationally inefficient in resolving localised flow features such as tip vortices.

Bottasso & Shephard [50] applied a finite element adaptive multigrid euler solver to rotary wing aerodynamics. They used an error indicator based on vorticity to adapt the mesh in the wake of the rotor. They discuss the issues with adaptive procedures with regards to new cell quality and recommend that the refined mesh be projected on the underlying geometry.

Abdel-Maksoud *et al.* [81] present experimental and numerical results for a propeller hub vortex. Different hub shapes are tested and compared with regards to efficiency and cavitation performance. A 1.3 million volume mesh was used with cells clustered near the hub vortex. The entire propeller was modelled in order to capture the root vortex interaction which would not have been possible with a cyclic boundary condition. The shear stress transport (SST) turbulence model is used and is stated that this turbulence model performs better especially in the separated regions but no evidence to support this is provided.

1.4 Summary and Layout

Tip vortices are important flow features which need to be identified and modelled correctly. From existing work it is apparent that a fine mesh is required in the region of the tip vortex. Non adaptive methods experience difficulties in having the required mesh density in the vortex core. An adaptive mesh refinement scheme can prove advantageous for such cases.

In Chapter 1 the basic details and existing work has been presented and discussed. In the Chapter 2 the theory of the numerical model used is presented. The Vortfind method is reviewed and refined in Chapter 3. The applicability of the method would be discussed and its advantages/capabilities presented. In Chapter 4 the method is extended to 3D and applied to a simple 3D wing to prove and validate the scheme.

An algorithm to capture thin shear wakes is developed in Chapter 5 and applied to a 3D wing in isolation and in conjunction with the VFX method. The mesh generation and tool development for marine propellers is reviewed in Chapter 6. The application of VFX to marine propellers is in Chapter 7 where the identification of complex vortex lines such as the helical propeller tip vortex is dealt with. Finally the method is applied to two marine propellers and compared with experimental results in Chapters 8&9.

2 Navier-Stokes Equations

To study vortical flows we need a numerical tool capable of modeling such problems. One such family of tools is based on the Navier Stokes equations. Exact analytical solutions to the Navier stokes equations exist only for a very limited number of flows. For real flows the equations have to be replaced by algebraic approximations, which have to be solved using an appropriate numerical method.

There are numerous programs to numerically solve the Navier Stokes equations with many different schemes and approximations. Each scheme has its advantages and disadvantages and the choice depends on what flow is going to be modeled. There are many commercial codes available which have been developed over the past years. They are robust and reliable with a wide range of mesh generation tools and utilities. Rather than reinvent the wheel, it was decided to use a commercial solver and concentrating the effort into the vortex identification scheme.

There were two commercial general purpose RANS solvers available at the School of Engineering Sciences at the University of Southampton. ***CFX*** by AEA Technologies [85] and ***Fluent*** by Fluent Inc [84]. Both are widely used in academia and commercial applications and have broadly similar capabilities. The codes were available on the university's computer facilities. Initial computations for the wings were carried out on Solaris, a SunFire VX880 with 6 processors. Later computations were performed on Iridis2, a large computational cluster consisting at the time of 300 Opteron processors. The cluster was expanded to 800 processors during the second phase of the installation.

CFX was chosen for this study for the following reasons. ***CFX*** has a strong marine following and the experience and knowledge was available in the Ship Science department. The second reason is that ***CFX*** was considered to be a more integrated package. The pre-processor, solver and post-processor are inter-linked allowing some useful features and functions. For example when the mesh is refined near a surface the new point is placed on the actual surface and not interpolated from the grid points resulting in a better representation of the geometry. In addition it was found from experience that the ***CFX*** coupled solver is more forgiving and robust concerning mesh quality. Fluent was found to have trouble solving what many consider reasonable quality grids for incompressible flows.

Incompressible turbulent flows are governed by the conservation laws for mass and momentum, the Navier-Stokes equations:

- The continuity equation simply states that the rate of change of mass in a control volume equals the rate of mass flux.
- The momentum equation states that the rate of change of momentum for the control volume is equal to the rate at which momentum is entering or leaving through the surface of the control volume, plus the sum of the forces acting on the volume.
- The energy equation states that the rate of change in internal energy in the control volume is equal to the rate at which enthalpy is entering, plus work done on the control volume by the viscous stresses.

Continuity equation in conservation form:

$$\frac{\partial \rho}{\partial t}+\nabla\bullet(\rho V)=0 \tag{1.1}$$

Momentum equations in conservation form:

$$\begin{aligned}
\frac{\partial(\rho u)}{\partial t}+\nabla\bullet(\rho uV)&=-\frac{\partial p}{\partial x}+\frac{\partial \tau_{xx}}{\partial x}+\frac{\partial \tau_{yx}}{\partial y}\frac{\partial \tau_{zx}}{\partial z}+\rho f_x\\
\frac{\partial(\rho v)}{\partial t}+\nabla\bullet(\rho vV)&=-\frac{\partial p}{\partial y}+\frac{\partial \tau_{xy}}{\partial x}+\frac{\partial \tau_{yy}}{\partial y}\frac{\partial \tau_{zy}}{\partial z}+\rho f_y\\
\frac{\partial(\rho w)}{\partial t}+\nabla\bullet(\rho wV)&=-\frac{\partial p}{\partial z}+\frac{\partial \tau_{xz}}{\partial x}+\frac{\partial \tau_{yz}}{\partial y}\frac{\partial \tau_{zz}}{\partial z}+\rho f_z
\end{aligned} \tag{1.2}$$

Energy equation in conservation form:

$$\begin{aligned}
&\frac{\partial}{\partial t}\left[\rho\left(e+\frac{V^2}{2}\right)\right]+\nabla\bullet\left[\rho\left(e+\frac{V^2}{2}\right)V\right]=\rho\dot{q}+\frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right)+\frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right)\\
&+\frac{\partial}{\partial z}\left(k\frac{\partial T}{\partial z}\right)-\frac{\partial(up)}{\partial x}-\frac{\partial(vp)}{\partial y}-\frac{\partial(wp)}{\partial z}+\frac{\partial(u\tau_{xx})}{\partial x}+\frac{\partial(u\tau_{yx})}{\partial y}+\frac{\partial(u\tau_{zx})}{\partial z}\\
&+\frac{\partial(v\tau_{xy})}{\partial x}+\frac{\partial(v\tau_{yy})}{\partial y}+\frac{\partial(v\tau_{zy})}{\partial z}+\frac{\partial(w\tau_{xz})}{\partial x}+\frac{\partial(w\tau_{yz})}{\partial y}+\frac{\partial(w\tau_{zz})}{\partial z}+\rho f.V
\end{aligned} \tag{1.3}$$

The Navier-Stokes equations cannot be solved analytically for all but a few cases. A numerical solution is sought for most cases. The computational effort to solve the complete Navier-Stokes equations is costly and for most engineering flows the equations are time averaged to get the Reynolds Averaged Navier-Stokes equations. The principle is that for steady flow the fluctuations in the flow are very small and a mean value is still valid.

Since the Navier-Stokes equations cannot be closed a turbulence model is required to allow the solution of the RANS equations. There are many such turbulence models available all of which have different advantages and disadvantages. An alternate more cost effective method to the direct numerical solution of the Navier-Stokes equations (DNS) is Large Eddy Simulation (LES) where the small scale turbulence is not modelled and only the larger turbulent flow features are accounted for. For more information see [82, 83].

2.1.1 Discretisation of the Governing Equations

The following approach is based on that used within CFX for which more details can be found in [85]. It is explained in order to explore some of the influences of the approach to the eventual solutions used later in this work. The approach involves discretising the spatial domain into finite control volumes to create what is called a mesh or grid. The governing equations are integrated over each control volume, such that the relevant quantity (mass, momentum, energy etc.) is conserved for each control volume.

The figure below shows a typical two dimensional mesh on which one surface of the finite volume is represented by the shaded area.

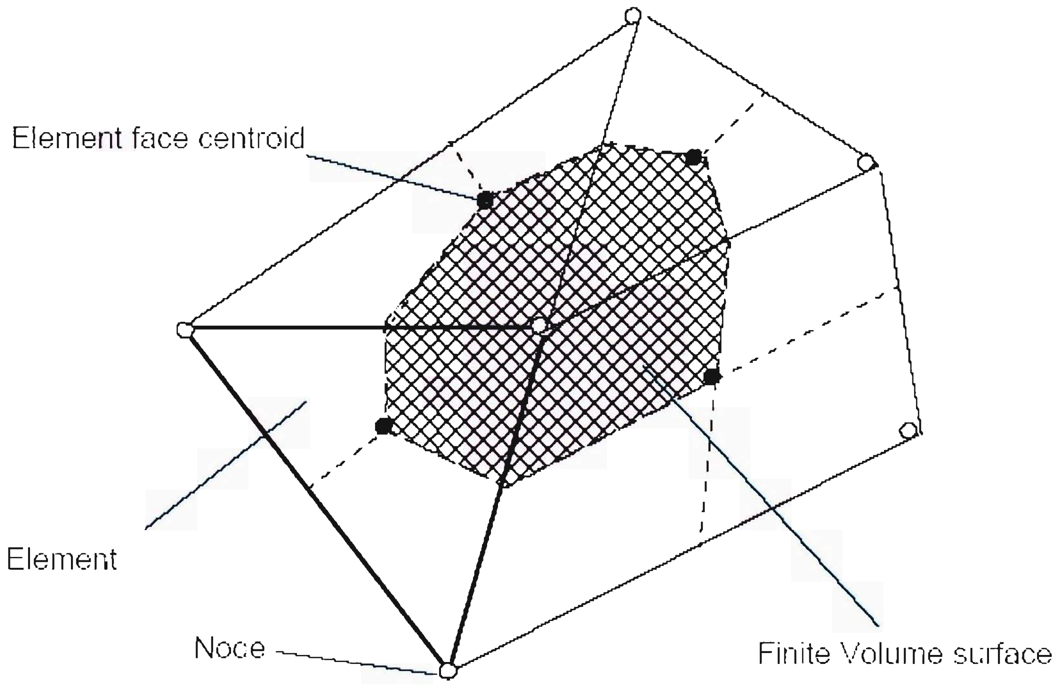


Figure 2.1 - Finite Volume Surface

It is clear that each node is surrounded by a set of surfaces which comprise the finite volume. All the solution variables and fluid properties are stored at the element nodes. Consider the mean form of the conservation equations for mass, momentum and energy, expressed in Cartesian coordinates:

$$\frac{\partial \rho}{\partial t}+\frac{\partial}{\partial x_j}\left(\rho U_j\right)=0$$

$$\frac{\partial}{\partial t}\left(\rho U_i\right)+\frac{\partial}{\partial x_j}\left(\rho \mu_j U_i\right)=-\frac{\partial P}{\partial x_i}+\frac{\partial}{\partial x_j}\left(\mu_{eff}\left(\frac{\partial U_i}{\partial x_j}+\frac{\partial U_j}{\partial x_i}\right)\right) \qquad (1.4)$$

$$\frac{\partial}{\partial t}\left(\rho \phi\right)+\frac{\partial}{\partial x_j}\left(\rho \mu_j \phi\right)=\frac{\partial}{\partial x_j}\left(\Gamma_{eff}\left(\frac{\partial \phi}{\partial x_j}\right)\right)+S_\phi$$

These equations can be integrated over a fixed control volume, using Gauss' divergence theorem to convert volume integrals to surface integrals as follows:

$$\frac{\partial}{\partial t}\int_v \rho dv+\int_s \rho U_j dn_j=0$$

$$\frac{\partial}{\partial t}\int_v \rho U_i dv+\int_s \rho \mu_j U_i dn_j=-\int_s P dn_j+\int_s \mu_{eff}\left(\frac{\partial U_i}{\partial x_j}+\frac{\partial U_j}{\partial x_i}\right)dn_j+\int_v S_{U_i} dv \qquad (1.5)$$

$$\frac{\partial}{\partial t}\int_v \rho \phi dv+\int_s \rho U_j \phi dn_j=\int_s \Gamma_{eff}\left(\frac{\partial \phi}{\partial x_j}\right)dn_j+\int_v S_\phi dv$$

where v and s denote volume and surface integrals respectively and dn_j are the differential Cartesian components of the outward normal surface vector. The surface integrals are the integrations of the fluxes, whereas the volume integrals represent source or accumulation terms.

The first step in solving these continuous equations numerically is to approximate them using discrete functions. Now consider an isolated mesh element such as the one shown in Figure 2.2.

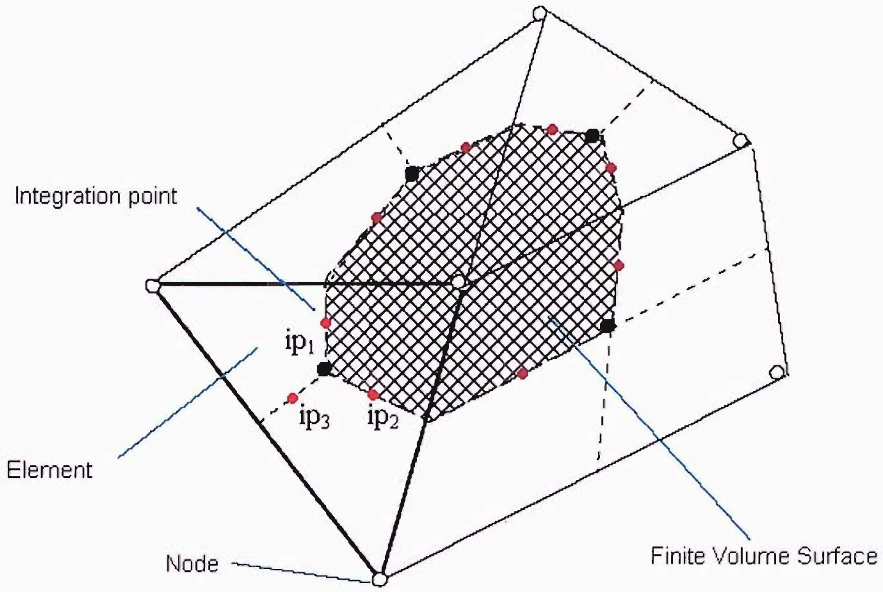


Figure 2.2 - Integration points

The surface fluxes must be discretely represented at the integration points to complete the conversion of the continuous equation into their discrete form. The integration points, ip_n, are located midway from the element face centroid to the element's sides (red dots in Figure 2.2). These integration points surround the finite volume if all adjacent face elements are considered (Figure 2.2).

The discrete form of the integral equations are written as:

$$\rho V\left(\frac{\rho-\rho^{o}}{\Delta t}\right)+\sum_{ip}\left(\rho U_j \Delta n_j\right)_{ip}=0$$

$$\rho V\left(\frac{U_i-U_i^{o}}{\Delta t}\right)+\sum_{ip}\dot{m}_{ip}\left(U_i\right)_{ip}=\sum_{ip}\left(P\Delta n_i\right)_{ip}+\sum_{ip}\left(\mu_{eff}\left(\frac{\partial U_i}{\partial x_j}+\frac{\partial U_j}{\partial x_i}\right)\Delta n_j\right)_{ip}+\overline{S_{U_i}}V \quad (1.6)$$

$$\rho V\left(\frac{\phi-\phi^{o}}{\Delta t}\right)+\sum_{ip}\dot{m}_{ip}\phi_{ip}=\sum_{ip}\left(\Gamma_{eff}\frac{\partial\phi}{\partial x_j}\Delta n_j\right)_{ip}+\overline{S_{\phi}}V$$

where V is the control volume, the subscript ip denotes an integration point, the summation is over all the integration points of the finite volume, Δn_j is the discrete outward surface vector, Δt is the timestep. For simplification a First Order Backward Euler scheme has been assumed in this equation, although a second order scheme was used in this work. Superscripts o refers to the old time level. The discrete mass flow through a surface of the finite volume is given by:

$$\dot{m}_{ip}=\left(\rho U_j \Delta n_j\right)_{ip}^{o} \qquad (1.7)$$

2.1.2 Pressure-Velocity Coupling

A single cell, unstaggered, and collocated grid is used to overcome the decoupling of pressure and/or velocity. The representation of mass conservation can be written as:

$$\left(\frac{\partial U}{\partial x}\right)_i + \frac{\Delta x^3 A}{4\dot{m}}\left(\frac{\partial^4 p}{\partial x^4}\right)_i = 0$$

where (1.8)

$$\dot{m} = \rho U_j \Delta n_j$$

The continuity equation is a second order central difference approximation to the first order derivative in velocity, modified by a fourth derivative in pressure which acts to redistribute the influence of the pressure. The method is similar to that used by Rhie and Chow [86], with a number of extensions which improve the robustness of the discretisation when the pressure varies rapidly, or is affected by body forces.

2.1.3 Diffusion Terms

Following the standard finite element approach, shape functions are used to evaluate the derivatives for all the diffusion terms. For example, for a derivative in the x direction at integration point *ip*,

$$\left.\frac{\partial \phi}{\partial x}\right|_{ip} = \sum_n \left.\frac{\partial N_n}{\partial x}\right|_{ip} \phi_n \qquad (1.9)$$

The summation is over all the shape functions for the element. The Cartesian derivatives of the shape functions can be expressed in terms of their local derivatives via the Jacobian transformation matrix:

$$\begin{bmatrix} \frac{\partial N}{\partial x} \\ \frac{\partial N}{\partial y} \\ \frac{\partial N}{\partial z} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} & \frac{\partial z}{\partial s} \\ \frac{\partial x}{\partial t} & \frac{\partial y}{\partial t} & \frac{\partial z}{\partial t} \\ \frac{\partial x}{\partial u} & \frac{\partial y}{\partial u} & \frac{\partial z}{\partial u} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial N}{\partial s} \\ \frac{\partial N}{\partial t} \\ \frac{\partial N}{\partial u} \end{bmatrix} \qquad (1.10)$$

The shape function gradients can be evaluated at the actual location of each integration point (true tri-linear interpolation), or at the location where each *ip* surface intersects the element edge (linear-linear interpolation).

2.1.4 Pressure Gradient Term

The surface integration of the pressure gradient in the momentum equations involves evaluation of the expression:

$$\left(P\Delta n_{ip}\right)_{ip} \tag{1.11}$$

The value of P_{ip} is evaluated using the shape functions:

$$P_{ip} = \sum_{n} N_n\left(s_{ip}, t_{ip}, U_{ip}\right) P_n \tag{1.12}$$

As with the diffusion terms, the shape function used to interpolate *P* can be evaluated at the actual location of each integration point (true trilinear interpolation), or at the location where each *ip* surface intersects the element edge (linear-linear interpolation).

2.1.5 Advection Term

To complete the discretisation of the advection term, the variable φ_{ip} must be related to the nodal values of φ. The advection schemes implemented in CFX-5 can be cast in the form:

$$\phi_{ip} = \phi_{up} + \beta\nabla\phi.\Delta\overset{\vee}{r} \tag{1.13}$$

where φ_{up} is the value at the upwind node, $\nabla\varphi$ is the gradient of φ and $\overset{\vee}{r}$ is the vector from the upwind node to the *ip*. Particular choices for β give rise to different schemes.

2.1.6 1st Order Upwind Differencing Scheme

A value of $\beta = 0$ leads to the first order Upwind Difference Scheme (UDS). UDS is very robust (numerically stable) and is guaranteed to not introduce non-physical overshoots and undershoots. However, it is also susceptible to a phenomenon known as Numerical Diffusion or 'gradient smearing' (see 2.1.14 Numerical Diffusion).

2.1.7 High Resolution Scheme

The High Resolution Scheme computes β locally to be as close to 1 as possible without violating boundedness principles. The recipe for β is based on that of Barth and Jesperson [87]. The high resolution scheme is therefore both accurate (reducing to first order near discontinuities and in the free stream where the solution has little variation) and bounded.

2.1.8 The Coupled System of Equations

The linear set of equations that arise by applying the Finite Volume Method to all elements in the domain are discrete conservation equations. The system of equations can be written in the form:

$$\sum_{nb_i} a_i^{nb} \phi_i = b_i \qquad (1.14)$$

where φ is the solution, *b* the right hand side, *a* the coefficients of the equation, *i* is the identifying number of the finite volume or node in question, and *nb* means "neighbour", but also includes the central coefficient multiplying the solution at the *i*th location. The node may have any number of such neighbours, so that the method is equally applicable to both structured and unstructured meshes. The set of these, for all finite volumes constitutes the whole linear equation system. For a scalar equation (e.g. enthalpy or turbulent kinetic energy), each a_i^{nb}, φ_{nb} and b_i is a single number. For the coupled, 3D mass-momentum equation set they are a (4 x 4) matrix or a (4 x 1) vector, which can be expressed as:

$$a_i^{nb} = \begin{bmatrix} a_{uu} & a_{uv} & a_{uw} & a_{up} \\ a_{vu} & a_{vv} & a_{vw} & a_{vp} \\ a_{wu} & a_{wv} & a_{ww} & a_{wp} \\ a_{pu} & a_{pv} & a_{pw} & a_{pp} \end{bmatrix}_i$$

and

$$\phi_i = \begin{bmatrix} u \\ v \\ w \\ p \end{bmatrix}_i \qquad (1.15)$$

$$b_i = \begin{bmatrix} b_u \\ b_v \\ b_w \\ b_p \end{bmatrix}_i$$

It is at the equation level that the coupling in question is retained and at no point are any of the rows of the matrix treated any differently (e.g. different solution algorithms for momentum versus mass). The advantages of such a coupled treatment over a non-coupled or segregated approach are several: robustness, efficiency, generality and simplicity. These advantages all combine to make the coupled solver an extremely powerful feature of any CFD code. The principal drawback is the high storage needed for all the coefficients.

2.1.9 Solution Method - The Coupled Solver

CFX-5 uses a coupled solver, which solves the hydrodynamic equations (for u, v, w, p) as a single system. This solution approach uses a fully implicit discretisation of the equations at any given time step. For steady state problems the time-step behaves like an 'acceleration parameter', to guide the approximate solutions in a physically based manner to a steady-state solution. This reduces the number of iterations required for convergence to a steady state, or to calculate the solution for each time step in a time dependent analysis.

2.1.10 General Solution

The flow chart shown below illustrates the general solution procedure.

The solution of each set of equations shown in the flow chart consists of two numerically intensive operations. For each timestep:

1. The non-linear equations are linearised (coefficient iteration) and assembled into the solution matrix.
2. The linear equations are solved (equation solution iteration) using an Algebraic Multigrid method.

The timestep iteration is controlled by the physical timestep (global) or local timestep factor (local) setting to advance the solution in time for a steady state simulation. In this case, there is only one linearisation (coefficient) iteration per timestep.

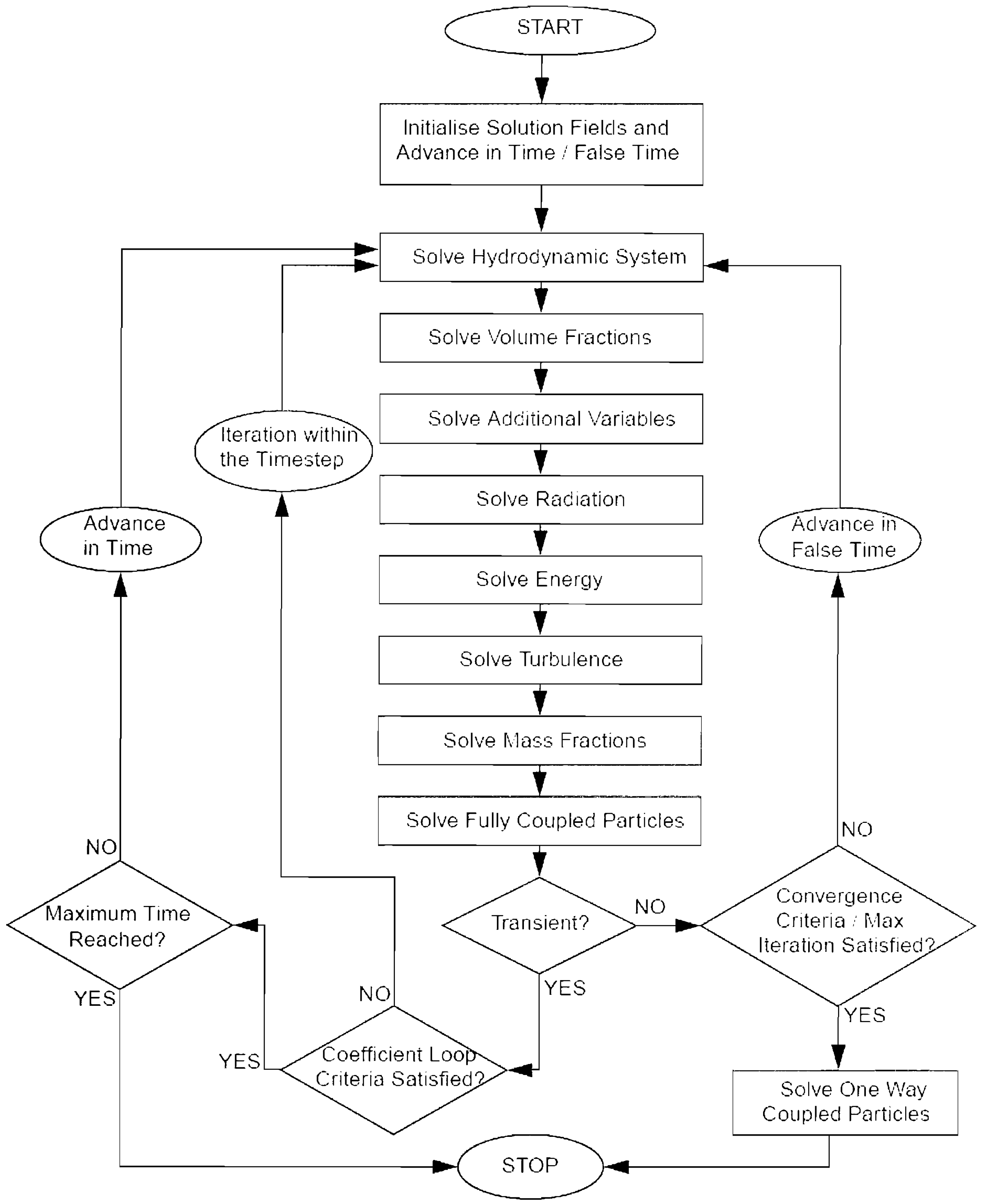

START
Initialise Solution Fields and Advance in Time / False Time
Solve Hydrodynamic System
Solve Volume Fractions
Solve Additional Variables
Solve Radiation
Solve Energy
Solve Turbulence
Solve Mass Fractions
Solve Fully Coupled Particles
Transient?
NO
YES
Convergence Criteria / Max Iteration Satisfied?
NO
YES
Advance in False Time
Solve One Way Coupled Particles
Coefficient Loop Criteria Satisfied?
NO
YES
Iteration within the Timestep
Maximum Time Reached?
NO
YES
Advance in Time
STOP

Figure 2.3 - Solution procedure [88]

2.1.11 Linear Equation Solution

A Multigrid (MG) accelerated Incomplete Lower Upper (ILU) factorisation technique is used for solving the discrete system of linearised equations. It is an iterative solver whereby the exact solution of the equations is approached during the course of several iterations.

The linearised system of discrete equations described above can be written in the general matrix form

$$[A][\phi]=[b] \tag{1.16}$$

where $[A]$ is the coefficient matrix, $[\varphi]$ the solution vector and $[b]$ the right hand side. The above equation can be solved iteratively by starting with an approximate solution, φ^{n}, that is to be improved by a correction, φ', to yield a better solution, φ^{n+1}, i.e.

$$\phi^{n+1}=\phi^{n}+\phi'$$

where ϕ' is a solution of

$$A\phi'=r^{n} \tag{1.17}$$

with r^{n}, the residual, obtained from,

$$r^{n}=b-A\phi^{n}$$

Repeated application of this algorithm will yield a solution of the desired accuracy.

By themselves, iterative solvers such as ILU tend to rapidly decrease in performance as the number of computational mesh elements increases. Performance also tends to rapidly decrease if there are large element aspect ratios present. The performance of the solver can be greatly improved by employing a technique called 'multigrid'.

2.1.12 Algebraic Multigrid

The convergence behaviour of many matrix inversion techniques can be enhanced by the use of a technique called 'multigrid'. The multigrid process involves carrying out early iterations on a fine mesh and later iterations on progressively coarser virtual ones. The results are then transferred back from the coarsest mesh to the original fine mesh.

From a numerical standpoint, the multigrid approach offers a significant advantage. For a given mesh size, iterative solvers are only efficient at reducing errors which have a wavelength of the order of the mesh spacing. So, while shorter wavelength errors disappear quite quickly, errors with longer wavelengths, of the order of the domain size, can take an extremely long time to disappear. The Multigrid Method bypasses this problem by using a series of coarse meshes such that longer wavelength errors appear as shorter wavelength errors relative to the mesh spacing. To prevent the need to mesh the geometry using a series of different mesh spacings, an Algebraic Multigrid is implemented.

Algebraic Multigrid [89] forms a system of discrete equations for a coarse mesh by summing the fine mesh equations. This results in virtual coarsening of the mesh spacing during the course of the iterations, and then re-refining the mesh to obtain an accurate solution. This technique significantly improves the convergence rates. Algebraic Multigrid is less expensive than other multigrid methods since discretisation of the non-linear equations is only performed once for the finest mesh.

CFX-5 uses a particular implementation of Algebraic Multigrid called Additive Correction [90]. The coarse mesh equations can be created by merging the original finite volumes to create larger ones as shown below. The diagram shows the merged coarse finite volume meshes to be regular, but in general their shape becomes very irregular. The coarse mesh equations thus impose conservation requirements over a larger volume and in so doing reduce the error components at longer wavelengths.

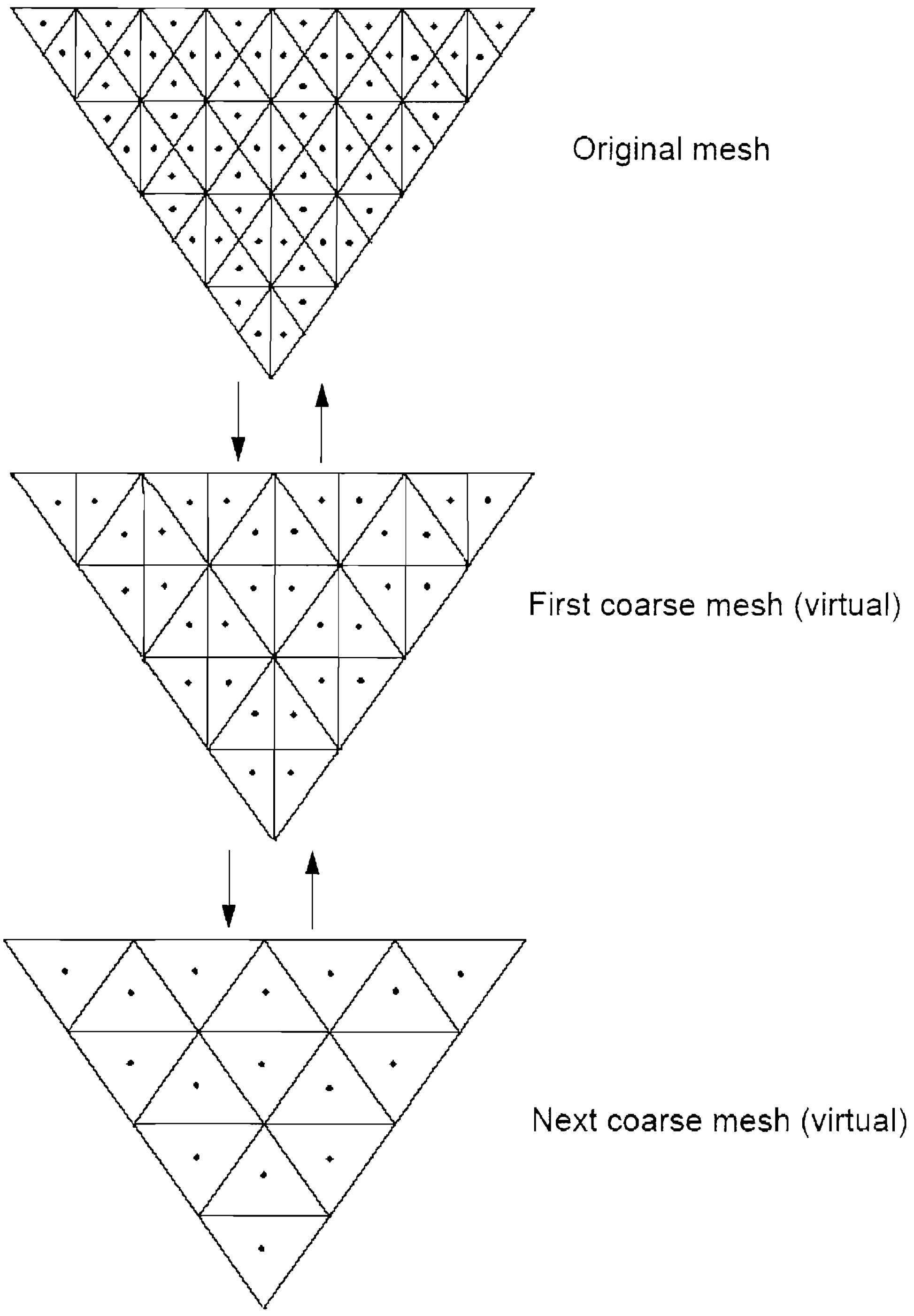


Figure 2.4 - Algebraic Multigrid

2.1.13 Discretisation Effects in CFX-5

All numerical approximation schemes are prone to a degree of error. Some errors are a result of truncation of additional terms in series expansions. Others are a result of the order of the differencing scheme used for the approximation.

Many of these effects can be significantly reduced or eliminated altogether by understanding why they occur, and when they are likely to affect the accuracy of the solution.

2.1.14 Numerical Diffusion

Numerical diffusion is an important issue when modelling vortical structures. Due to the re-circulating nature of the vortex it is very hard to avoid numerical diffusion. Numerical diffusion is usually exhibited by difference equations where the advection term has been approximated using an odd-order scheme, for instance, UDS, which is first order accurate.

Consider a 3-dimensional Cartesian coordinate system. On a mesh of quadrilateral elements, the flow direction may be normal to the faces of each element. In this case, the flow from one element to the next can be accurately represented to the limit of the mesh size.

In a case where the flow is not normal to the faces of the elements, perhaps in a region where the flow is re-circulating, the flow must move from one element into more than one element downstream. Consequently, some flow moves into each of the adjacent elements as shown below.

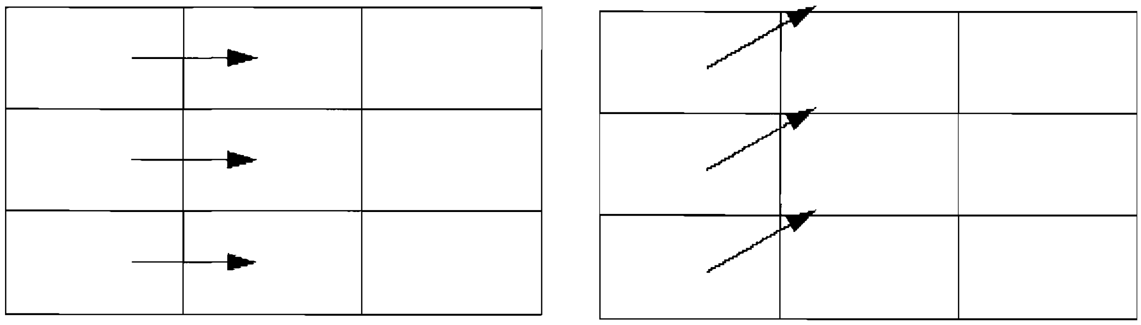


Figure 2.5 - Flow that is not normal to the cell faces causes numerical diffusion

The effect of this over a whole flow domain is that the features of the flow are smeared out. The diagram below illustrates the effect. If a step function is used to define the inlet profile but is not aligned with the mesh, the step is progressively smeared out as flow moves through the domain. This phenomenon is therefore sometimes called ‘gradient smearing’.

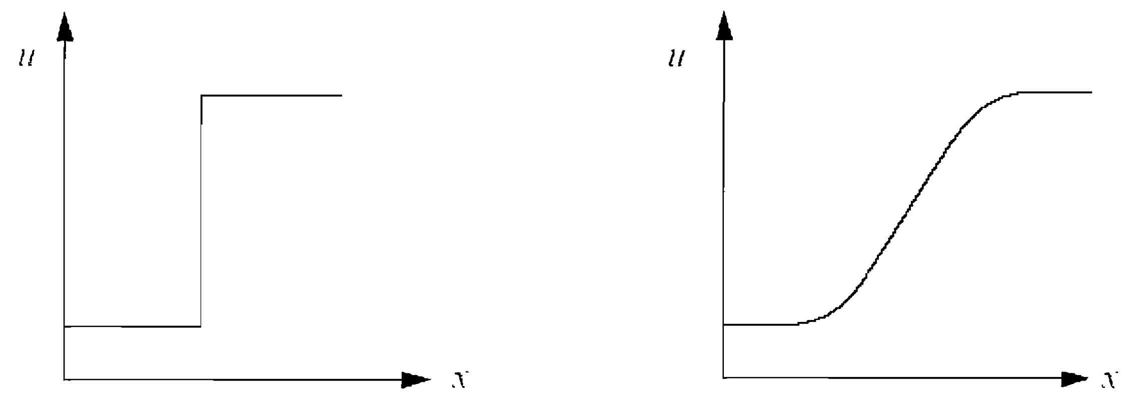


Figure 2.6 - Numerical diffusion

The effect varies according to the alignment of the mesh with the flow direction. It is therefore relatively straightforward to achieve highly accurate solutions to simple flow problems, such as flow in a duct where alignment of the mesh with the predominant flow is relatively simple. However, for situations in which the flow is predominantly not aligned with the mesh, numerical diffusion effects limit the accuracy of the solution.

Consider a similar flow, modelled on a totally unstructured tetrahedral mesh, as shown below. With this type of mesh, there is no flow direction which is more or less prone to numerical diffusion than any other. Consequently, the inaccuracy for simple unidirectional flows is greater than for a mesh of hexahedral elements aligned with the flow. However, the numerical diffusion errors for a mesh of tetrahedra are consistent, and of the same order, throughout the flow domain. This means that for real flows, tetrahedral control volumes will not exhibit additional inaccuracies in areas such as recirculation, because there is no single flow direction which may be aligned with the mesh.

It is a fact that using the UDS scheme with tetrahedral element meshes will produce solutions that exhibit a larger degree of numerical diffusion than would exist from a solution obtained with a similarly refined mesh of hexahedral elements. However, this discrepancy diminishes rapidly as the advective discretisation is made more second-order accurate, and by working towards a grid independent solution.

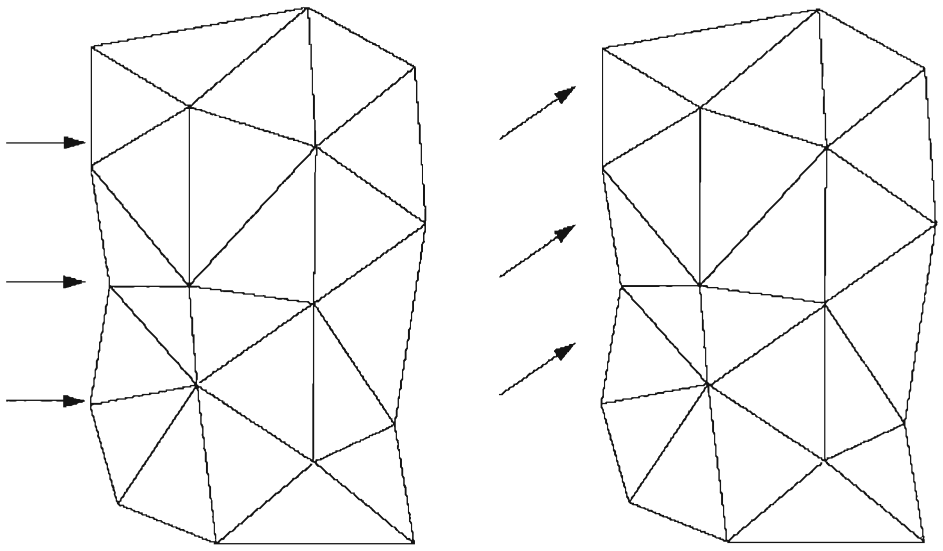


Figure 2.7 - Flow direction is trivial on unstructured grids

It is almost impossible to create a structured grid that minimises diffusion within a vortex. In order for the grid to accomplish this an O-grid inside the vortex must be created that has its radial faces perpendicular to the vortical flow (Figure 2.8). It is practically impossible to achieve this without solving the flow first. A slight offset in the vortex core relative to the grid centre will make such a grid pointless. In addition not all vortices are perfectly circular in shape which complicates things even further.

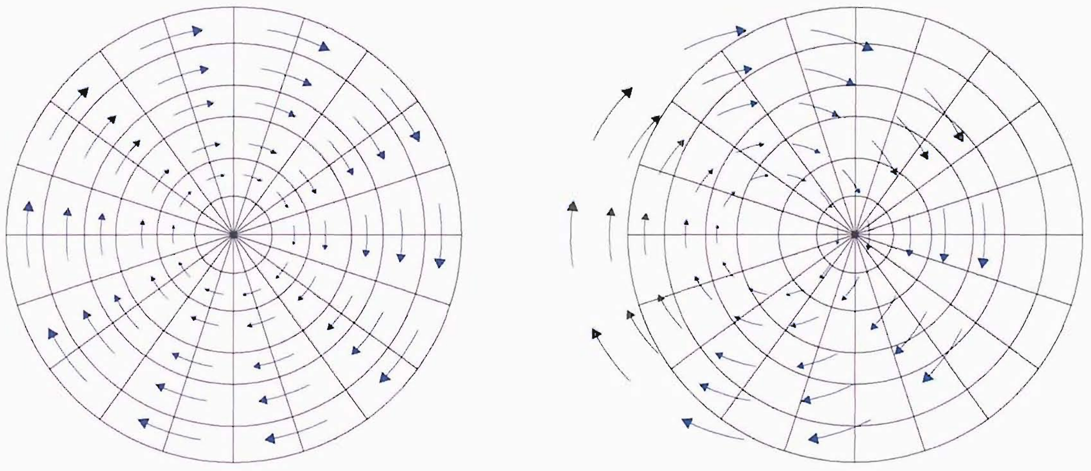


Figure 2.8 – A small shift in the vortex position makes any advantages of an O-grid obsolete

If the true three dimensional nature of the tip vortex is considered then the ideal mesh would also have a helical pattern to it. This is because the tip vortex has a stream wise component as well which mean that in order for the cell faces to be perpendicular to the flow a helical structure is required. The only advantage of have a hexahedral mesh is that the longitudinal mesh size can be bigger thus decreasing the mesh size.

As shown above an unstructured grid will have greater numerical diffusion than an aligned structured grid, but for vortical flows this disadvantage disappears quickly since the structured grid is no longer aligned.

2.1.15 Numerical Dispersion

Numerical dispersion is usually exhibited by discretised equations whose advection term has been approximated using schemes that are even-order accurate. When Numerical Advection Correction is fully implemented with a value of β=1.0 the scheme is second-order accurate. This can lead, in some cases, to numerical dispersion.

Dispersion results in oscillations or 'wiggles' in the solution particularly where there are steep flow gradients. Again the effects can be illustrated using the step function as shown in the diagram below; just before and just after the step, the solution exhibits oscillations which are the direct result of numerical dispersion.

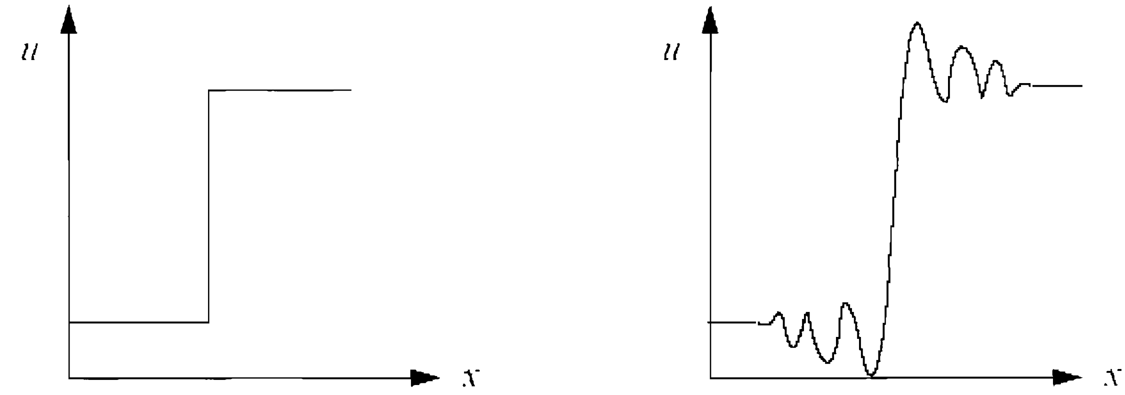


Figure 2.9 - Numerical dispersion

2.2 Turbulence modelling

Turbulence modelling has a major influence on the accuracy of vortex flows. Many of the existing numerical simulations have difficulty capturing the vortex radius correctly and this is frequently attributed to limitations associated with the turbulence model. There have been many studies comparing the performance of different turbulence models. For example, Osama *et al.* [91] compared the performance of Baldwin-Lomax, Spallart-Allmaras and k-ω turbulence models. All of the models over predicted the radius of the vortex with the $k\omega$ performing the best out of the models tested in the far-field region but more poorly near the wing surface.

Dacles-Mariani *et al.* [41] used a Baldwin-Lomax and a modified Baldwin-Lomax turbulence model with promising results. Even though the vortex core velocity profile was predicted within 3% of the experimental data the vortex decay just outside the core was not correct, with the vortex having an influence over the flow twice the distance as compared with the experimental results.

Wallin and Girimaji [92] investigated the effect of turbulence model on axial vortex decay rate. They used several turbulence models from Reynolds stress transport to eddy-viscosity k-ε models. The k-ε models over predicted the vortex decay rate with Reynolds stress transport models giving better results.

An increasing amount of research is being carried out on large eddy simulation (LES) and detached eddy simulation (DES). These methods address the physics of turbulence directly and do not require turbulence closure approximations. They require very fine meshes and are thus computationally expensive as demonstrated by Arakawa *et al* [95] when modelling a wind turbine blade tip using the Earth Simulator. DES only applies this methodology in certain regions of the flow. If an adaptive grid based on vortex identification is used then DES can be applied in the refined region to model a tip vortex.

For this research the k-ε model was used for the development of the VFX method because of its speed, simplicity and robustness. The 2 equation SST model by Menter [96] was used for subsequent simulations for its improved performance in regions for separated flow.

2.3 In-viscid flow (Euler)

Far away from solid boundaries the effects of viscosity are usually small. If viscous effects are neglected, the Navier-Stokes equations reduce to the Euler equations. Euler equations are useful for high Reynolds number problems where the effects of viscosity are usually confined to a small region near the body and a narrow wake. Since the boundary layer near the solid surfaces does not have to be resolved a coarser grid can be used which reduces computational costs and allows for more complex geometries. For more information see [82, 83].

3 VORTFIND scheme

The VORTFIND scheme [1, 2] is a method for identifying vortices in a two-dimensional velocity field. It was developed by Pemberton [1] for his thesis and applied to a few test cases such as a backward facing step and a 2-D bilge vortex. Later as part of this research the method was applied to a 3-D system of bilge vortices [97, 98]. In this chapter the VORTFIND method is presented and the key parameters investigated by applying it to a test case. The method is refined and extended before it is extended to 3-D in the next chapter.

The definition of a vortex as described by Lugt [36] "*A vortex is the rotating motion of a multitude of material particles around a common centre*" is used in the context of this method. The VORTFIND scheme is based on a simple function of local angles of velocity with respect to a reference point in the fluid. This function exhibits a local minimum at the vortex core. A statistical method can then be applied to locate the vortex core. It has been applied in two-dimensional velocity field with good results [1, 2].

3.1 Numerics

Consider a two-dimensional slice of fluid perpendicular to the axis of rotation of the vortex. A cell centre is selected as the datum point. The x-axis is used as a reference and the plane is divided into *n* sectors and each sector is assigned an integer value *β*, (Figure 3.1).

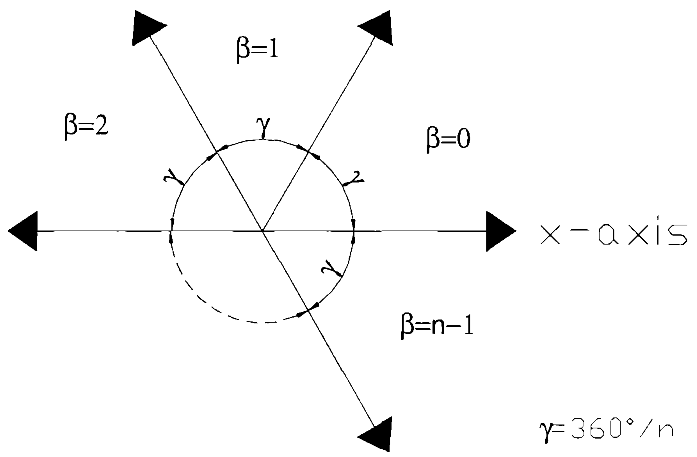


Figure 3.1 - The plane is split into sectors and each sector is assigned a value.

The angle α (Figure 3.2), which is the angle the velocity vector makes with the reference axis (in this case x-axis), is calculated for all data points. Each data point is then assigned a value β depending on which sector α lies in. The closest point to the reference location for each value of β is found. These points have distances labelled r_0, r_1, ..., r_n and $\beta = 0, 1, ..., n-1$ respectively (Figure 3.2). The distance for the value of β that is the same as the reference point will always be zero. Once the distances are found the l function is computed as follows:

$$l = \sum_{0}^{n-1} r_n^2 \tag{1.18}$$

Referring back to Lugt's definition, a vortex core is the point that is closest to points with differing values of β. At this point the l function exhibits a local minimum.

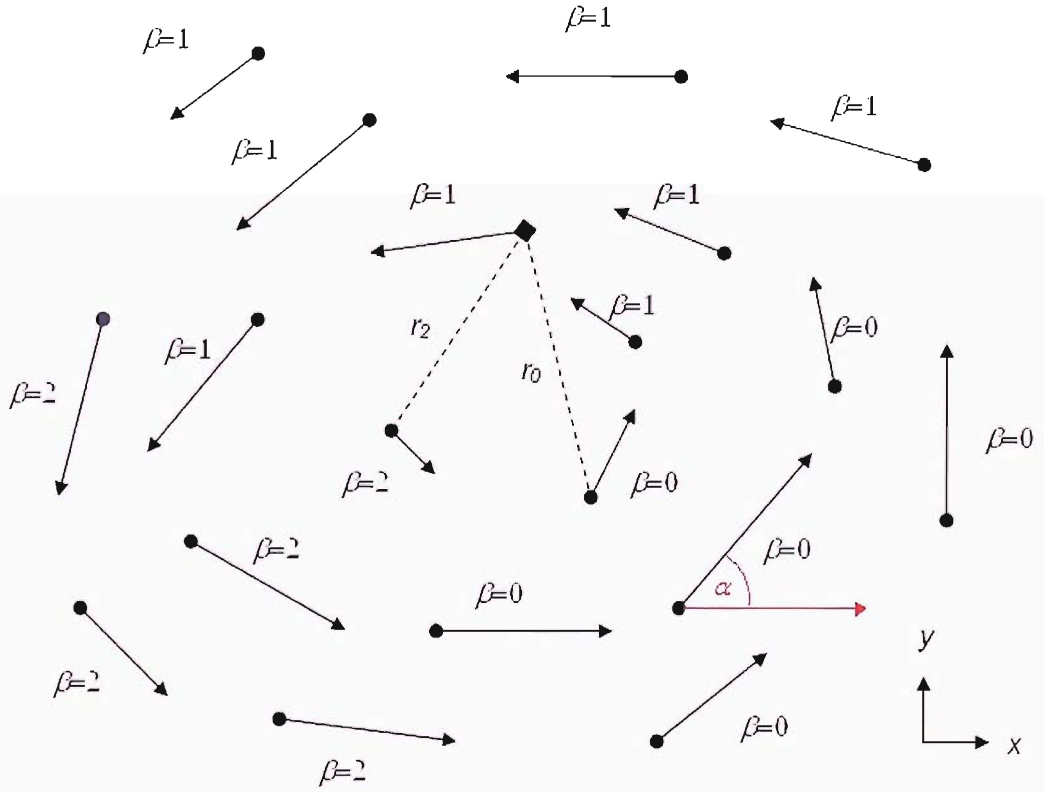


Figure 3.2 – Schematic representation for determining the l function for 3 sectors. The reference point is marked with a diamond

3.2 Identifying the vortex core

The *l* function provides a useful picture of the vortex structures in the flow. It can be used for adaptive refinement. However it is sometimes easier to adapt to the vortex core line. In addition for visualisation purposes it is better to identify the vortex core line [62]. The process of locating multiple vortex cores in the domain is one of identifying local minima in the *l* function. The search method has a number of constraints [1]:

- The *l* function is calculated without the use of any gradient values and it is beneficial to use a search algorithm that refrains from doing so.
- Multiple local minima may exist which are all significant, especially for adaptation.
- The search algorithm must have minimal computational requirements if is going to be included in the solution process.

Given the above constraints Pemberton [1] found that a K-Means cluster algorithm is appropriate for this case. The objective of a cluster algorithm is to separate a set of data into clusters so that the members of each cluster differ as little as possible with respect to a specified criterion [99]. The algorithm used is an adaptive K-Means Algorithm [100]. Data points are assigned to clusters by minimizing *J*, the sum of the distances squared from the cluster centre to the points within it (Eq.(1.19)).

$$J = \sum_{j=1,i=1}^{j=N_{pi},i=N_c} \left(x_{ji} - \mu_{xi}\right)^2 + \left(y_{ji} - \mu_{yi}\right)^2$$

$$\mu_{xi} = \frac{1}{N_{pi}} \sum_{j=1}^{N_{pi}} x_{ji} \tag{1.19}$$

$$\mu_{yi} = \frac{1}{N_{pi}} \sum_{j=1}^{N_{pi}} y_{ji}$$

where μ_{xi} and μ_{yi} are co-ordinates of the cluster centre, N_{pi} is the number of points in the ith cluster, N_c is the number of clusters and x_{ji}, y_{ji} are the co-ordinates of the jth point in the ith cluster. The points are assigned to the different clusters until *J* is minimized. Only two variables need to be preset in the K-Means cluster algorithm, the maximum cluster radius and the minimum separation between adjacent clusters. If a point is further away than the maximum cluster radius from a cluster then a new cluster is created. If two clusters are closer than the minimum separation then they are merged.

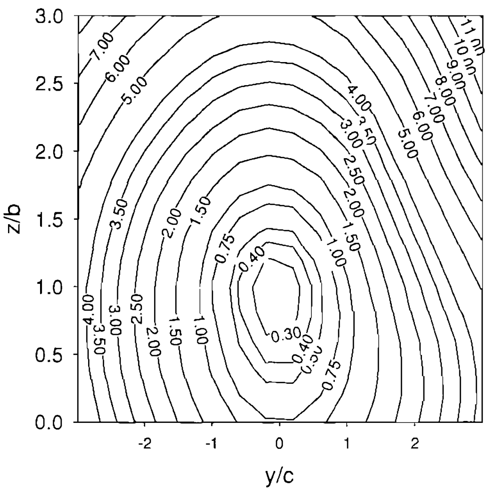
For the case of a single vortex on the computational plane it has been found that the node with the lowest *l* function is within a cell of the vortex core. Thus for single vortices identifying the vortex core is trivial.

3.2.1 VORTFIND Test case

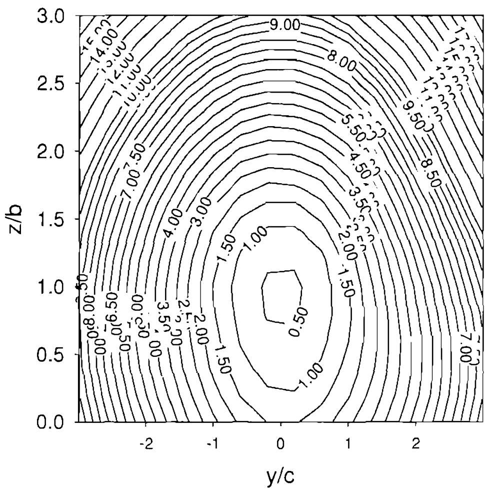
The VORTFIND scheme has not been applied to a tip vortex before. To test the applicability of the method to tip vortex flows, it was applied to a wing operating at an angle of attack. A 2-D plane that includes the tip vortex was used for the study.

A NACA0020 wing with a 1.0m span and 0.667m chord was used as the test case of the VORTFIND method in 2-D. The grid was unstructured with 250,000 tetrahedral cells. A k-ε turbulence model was used and the chosen angle of attack was 10^o. A plane 0.13m behind the wing trailing edge was used to calculate the *l* function. The velocities are exported from the volume mesh at the points where a cell edge intersects with the plane.

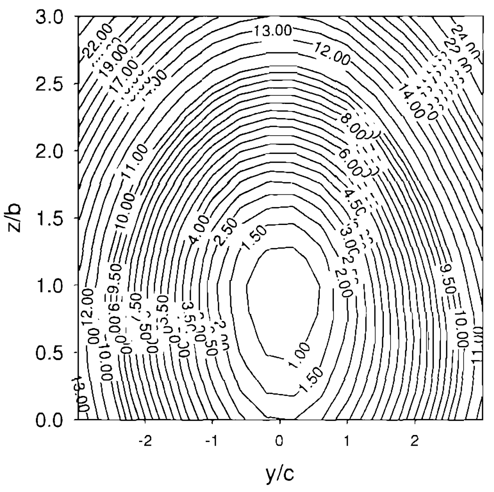
The effect of increasing the number of sectors used to calculate the *l* function can be seen in Figure 3.3. As the number of sectors increases the *l* function gets smoother. Above 5 sectors there is no longer significant difference in the *l* function.



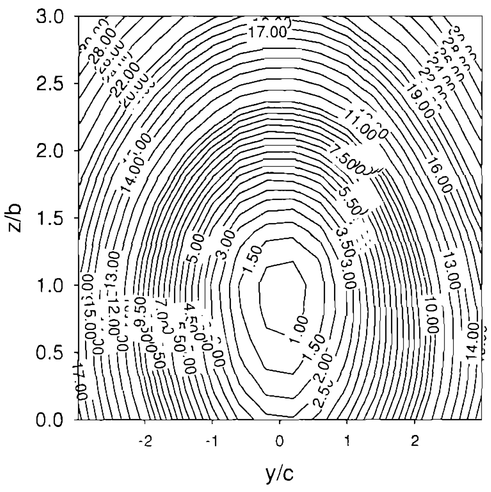
Sectors = 3



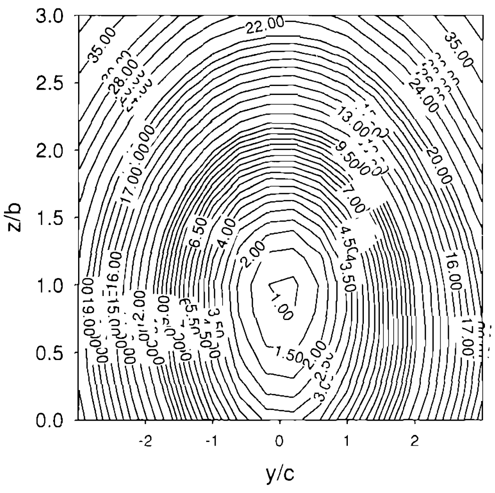
Sectors = 4



Sectors = 5



Sectors = 6



Sectors = 7

Figure 3.3 - *l* function contour plot for a plane 0.2c downstream of the trailing edge.

As the number of sectors increases from 3 to 7, the *l* function varies in magnitude since extra r_n are added for each sector.

	y/c	z/b
Wing tip	0.000	1.000
Sectors = 3	0.069	0.891
Sectors = 4	0.066	0.899
Sectors = 5	0.061	0.886
Sectors = 6	0.030	0.835
Sectors = 7	0.039	0.848

Table 3-1 - Position of Vortex centre

Another thing to note from the *l* function plots is that as the number of sectors increases the *l* function increases since one more distance is added for each additional sector (Eq(1.18)). This can be a problem when using the *l* function with adaptive refinement grids. If the grid is refined using a threshold *l* function value it is difficult to choose the correct one and is more of a trial and error [69]. If the *l* function changes with the number of sectors then the process is complicated further.

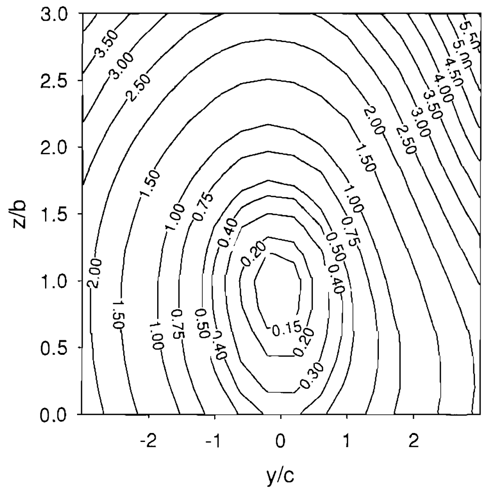
By dividing the *l* function by the number of sectors that exist in the solution minus one, the *l* function can be normalised with respect to sector number.

$$l_n = \frac{\sum_{0}^{n-1} r_n^2}{n_{\beta\_exist} - 1} \qquad (1.20)$$

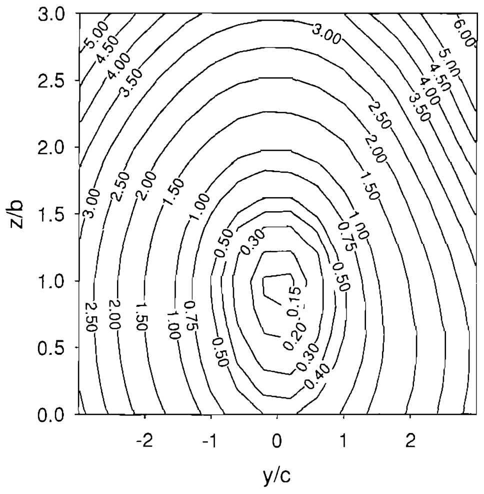
where n_{β_exist} is the number of sectors that have at least on velocity vector.

The reason for not dividing by the number of sectors used is that in some flows not every sector has a velocity vector assigned to it, so the distance squared for that sector is not added to the *l* function. Also the sector of the reference grid point always has a distance squared of zero so the *l* function is divided by the number of sectors that have at least one vector minus one. The l_n function is calculated for the same case as previously and shown in Figure 3.4.

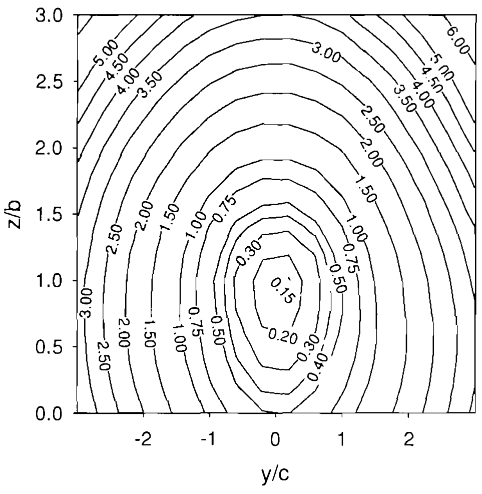
The above has no effect on the shape of the contours of the *l* function or the *K* means cluster algorithm (Table 3-1&Table 3-2). The only difference is that the values remain fairly constant with increasing number of sectors, especially near the vortex centre. If the l_n function contour of 0.5 is observed, its position does not change with increasing sectors. The same is not true for the *l* function. This simplifies the adaptation process if a threshold value scheme is used.



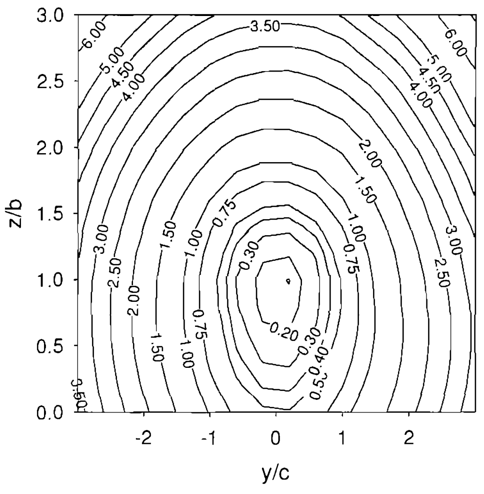
Sectors = 3



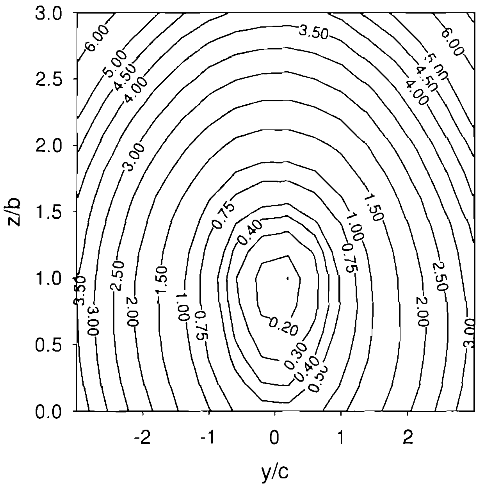
Sectors = 4



Sectors = 5



Sectors = 6



Sectors = 7

Figure 3.4 - Normalised l_n function contour plot for a plane 0.2c downstream of the trailing edge.

As the number of sectors increases from 3 to 7, the l function remains fairly constant in magnitude.

	y/c	z/b
Wing tip	0.000	1.000
Sectors = 3	0.069	0.891
Sectors = 4	0.066	0.899
Sectors = 5	0.061	0.886
Sectors = 6	0.030	0.835
Sectors = 7	0.039	0.848

Table 3-2 - Position of Vortex centre.
Normalised *l* function

3.2.2 Influence of grid on VORTFIND method

The l_n function is derived from the velocity vectors of neighbouring data points. It is dependant on the direction of the vectors belonging to different sectors. However the minimum possible l_n function is only dependent on the sampling spacing. Consider a reference point with all its neighbouring points belonging to different sectors. Obviously this point should exhibit the lowest l_n function. How low the data point's l_n function can be is decided by the grid spacing. If the data points are very close then the small distances between them will result in a low l_n function. If the points are far apart then the l_n function will be higher even though the neighbouring values are all in different sectors. A refined area can have a very low minimum l_n function whereas a coarse area has a higher minimum l_n function.

Compare the l_n function plot for the wing tip vortex with uniform sampling spacing Figure 3.5, with the l_n function for the same results but with samples taken at the cell edges Figure 3.6. We can see that away from the vortex core the l_n values are very similar. This is because the limiting factor for the l_n function is not the sample spacing but the actual velocity flow field. As we approach the vortex core the values for the uniform sampling spacing are higher. This is due to the smaller sample spacing near the vortex core. Since the flow field is more varied in direction the limiting factor becomes the sample spacing. Ideally the limiting factor should always be the flow field and never the sampling space.

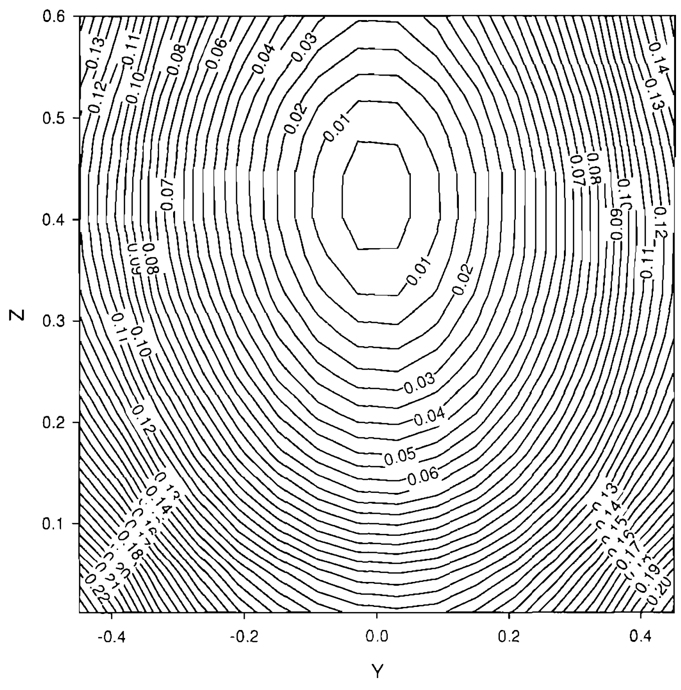


Figure 3.5 – Uniform spacing

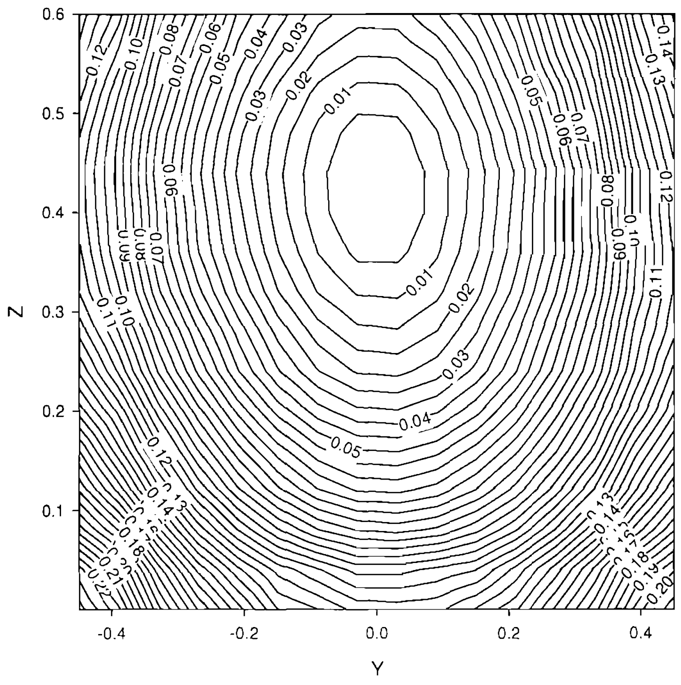


Figure 3.6 – Non-uniform spacing

However this is not always practical due to the increased computational overhead of such a fine sampling spacing. A uniform sampling spacing can overcome some of the problems of having the sampling spacing as a limiting factor. Effectively there is a lower limit on the l_n function which is the same for the entire computational plane. Using a non-uniform sampling spacing will potentially favour certain regions which may cause problems in flows with multiple vortices.

3.2.3 VORTFIND not conforming to computational nodes

The l_n function so far has been calculated at locations corresponding to computational nodes (i.e. locations were the velocity is specified). However the l_n function can be calculated at any point along the plane and does not need to conform to those points. The test case data was used to calculate the l_n function over a uniform 51x51 grid superimposed on the data on the nodes resulting from the mesh. The l_n function is smooth and continuous over the computational plane. The magnitude and shape is virtually the same as the one calculated on the data nodes. This is a significant advantage over the combinatorial method [65] which is discontinuous.

Even though the VORTFIND method is accurate enough to locate the cell the vortex core lies in, on large grid spacing it is thus possible to find the point of minimum l_n function within the identified cell using this not conforming procedure. Even though the physical meaning of this point is dubious, it is nevertheless a useful improved estimate on the position of the vortex core.

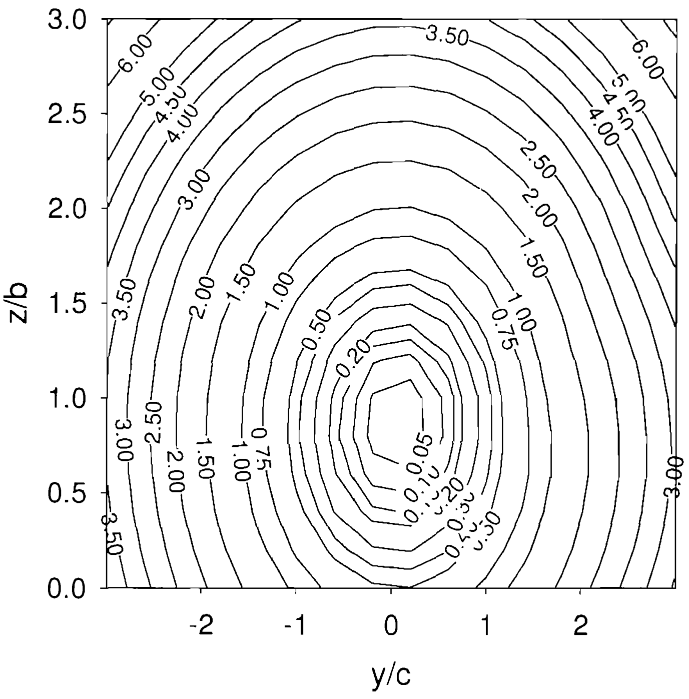
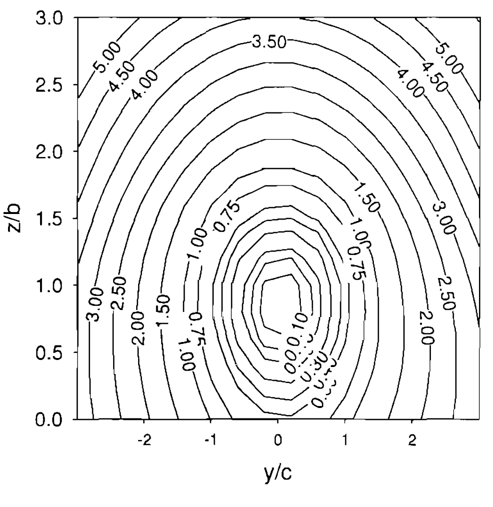


Figure 3.7 – l_n function off the data points. 7 sectors (compare to Figure 3.4)

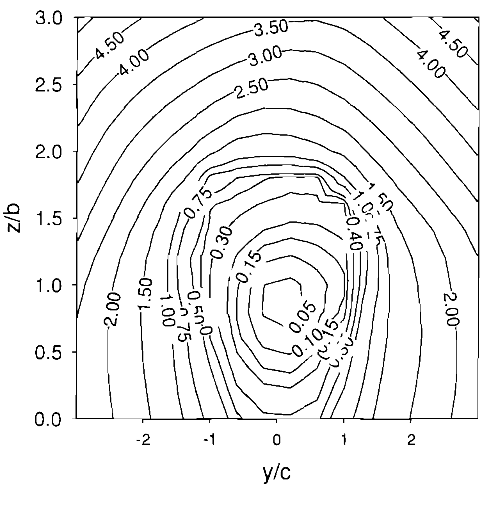
3.2.4 VORTFIND on coarse grids

The minimum possible l_n function as explained previously depends on the grid spacing. However there is another consideration concerning grid size. There is a limit to how coarse a grid can be in order to identify the vortex. A series of grids has been set up to investigate the lower limit of this coarseness. The initial grid is the same as all the other test cases. The nodes are then decreased by a factor of 4, 8 and 16 and the VORTFIND method carried out.

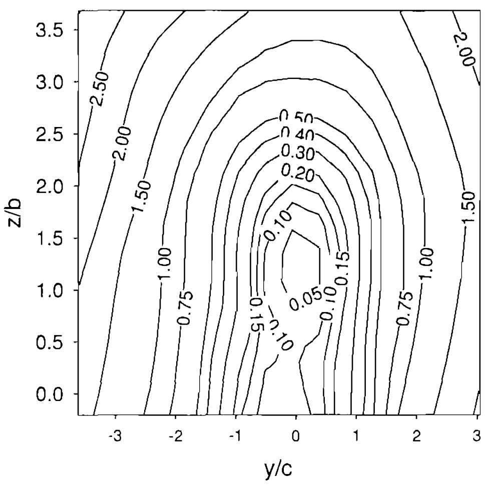
Inspecting the resulting l_n function contour plots we can see that we get similar shapes for all node densities with similar minimum values in the vortex core. Assuming the lowest l_n function as the vortex core there is a variation in the identified centre (Table 3-3). This is due to the way the data node grid was coarsened. Alternate nodes were deleted from the database which was not given in any particular order. This resulted in an uneven coarsening of the data grid nodes which leads to the shift in the vortex centre. However the shape of the l_n function over the sample plane remains similar even for the lowest of data node densities. This is a key result as it indicates one of the main advantages of the use of Vortfind as it still works well for a very coarse mesh.



3038 nodes



760 nodes


380 nodes

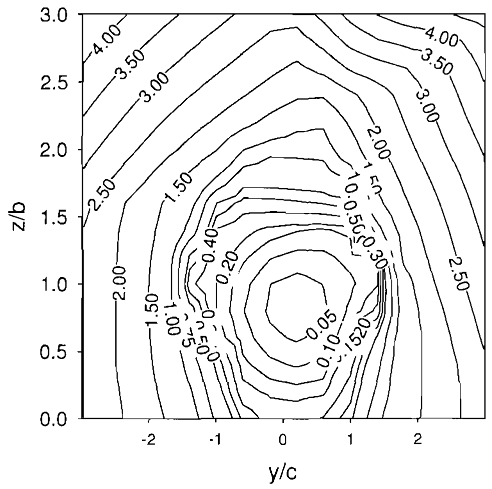

190 nodes

Figure 3.8 – l_n function on different grid densities

Number of data nodes	Identified vortex centre	
	y/c	z/b
3038	0.039	0.848
760	0.285	0.848
380	0.069	1.123
190	0.171	0.848

Table 3-3 - Variation of vortex core with grid density

The same procedure was carried out for the same data nodes using the l_n function not conforming to the data nodes. The grid on which the l_n function was calculated was 51x51. Similar results can be observed. The contour plots have the same characteristic shape. Similar variations in the identified vortex core position exist (Table 3-4). This is because as explained these are due to the data nodes. Since the data nodes are identical for both cases similar results were observed. Comparing the two methods we can see very good agreement considering the resolution for the 51x51 grid is in the order of 0.05b and 0.12c. A marked improvement over the conforming approach is evident from the identified vortex centres.

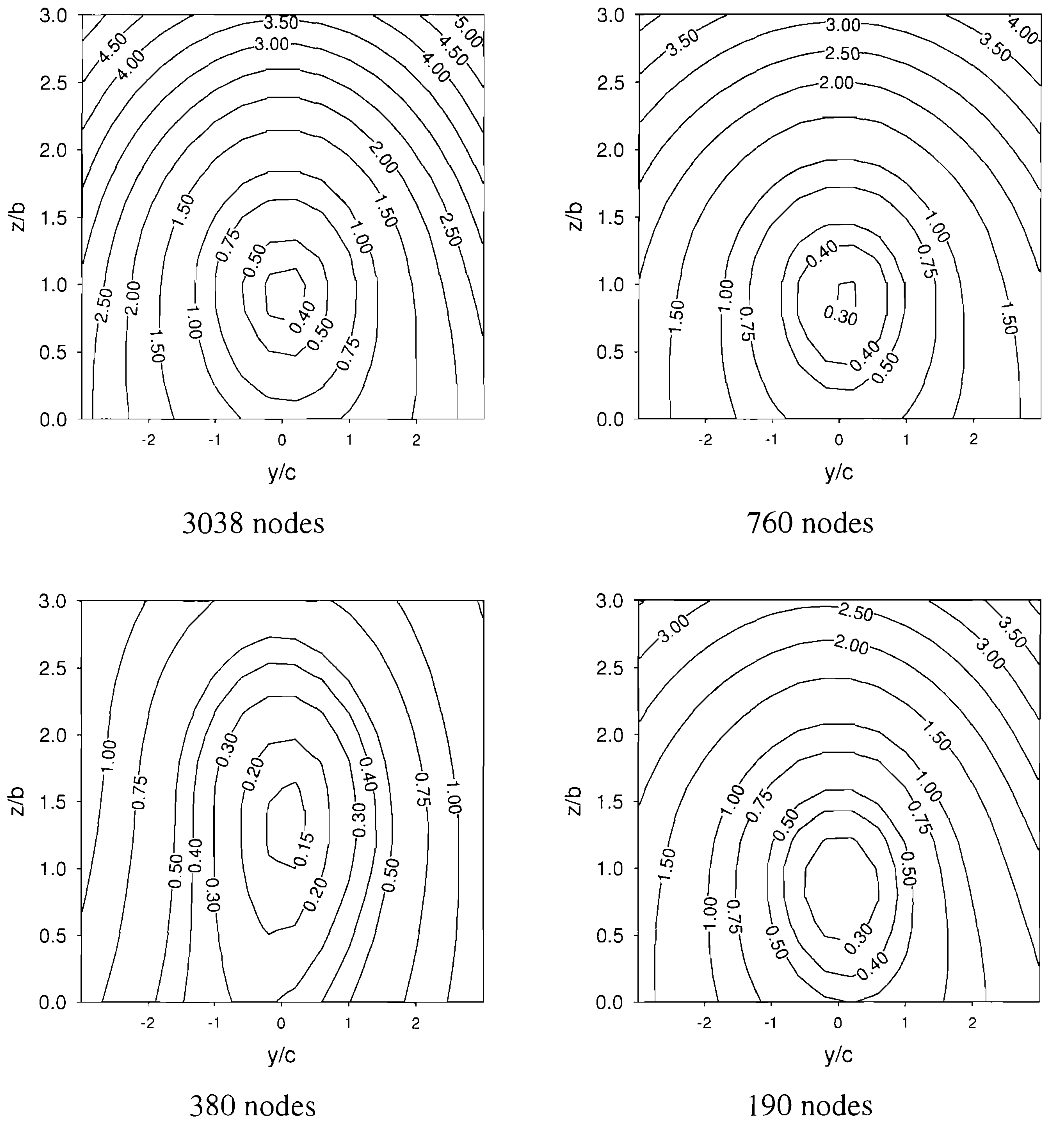


Figure 3.9 – l_n function on different grid densities not conforming to data nodes

Number of data nodes	Identified vortex	
	y/c	z/b
3038	0.08	0.90
760	0.08	0.90
380	0.08	1.14
190	0.16	0.90

Table 3-4 - Variation of vortex core with grid density (VFX not on data nodes)

3.3 Summary

In this chapter the Vortfind method has been presented and tested. The method was applied to a test case and its dependency on different parameters investigated. The ability of the Vortfind method to perform well on very coarse grids has been demonstrated, which is one of its major advantages.

In addition the Vortfind method has been reformulated for an arbitrary number of sectors and has been normalised with respect to these sectors. This makes the Vortfind method easier to implement in conjunction with mesh adaptation based on threshold values as explained before.

The method has also been applied for the first time off the data nodes which has proven that the l_n function is a continuous function over the plane. It performs particularly well on coarse grids and has the potential to identify the vortex core at a resolution which is better than the data grid.

4 VFX: an extension of VORTFIND to 3-D

The modified VORTFIND scheme can be easily implemented with two-dimensional grids. However, before this work it has never been applied to three-dimensional grids. A number of modifications must be made and evaluated before the method can be used with any success. The method works in two-dimensional planes perpendicular or near to perpendicular to the direction of the vortex core. These planes can be extracted from the three-dimensional velocity field (Figure 4.1).

The planes can be extracted for every cell or for a predetermined spacing. Extracting planes for every cell is computationally expensive; however since the l function is defined at every cell it can be used as a criterion for adaptive refinement without any further manipulation. A threshold value for the l function can be set and the cell below that threshold refined. This eliminates the need for a cluster algorithm to identify the vortex cores.

Using spaced planes reduces the computational requirement substantially, but means that the vortex cores must be identified and a vortex core line constructed through the domain. Then neighbouring cells to the vortex core line can be flagged for refinement. The other advantage of this method is that it produces a continuous vortex core line. The latter was chosen for its reduced computational requirements and adapted for 3-D cases giving the VFX scheme.

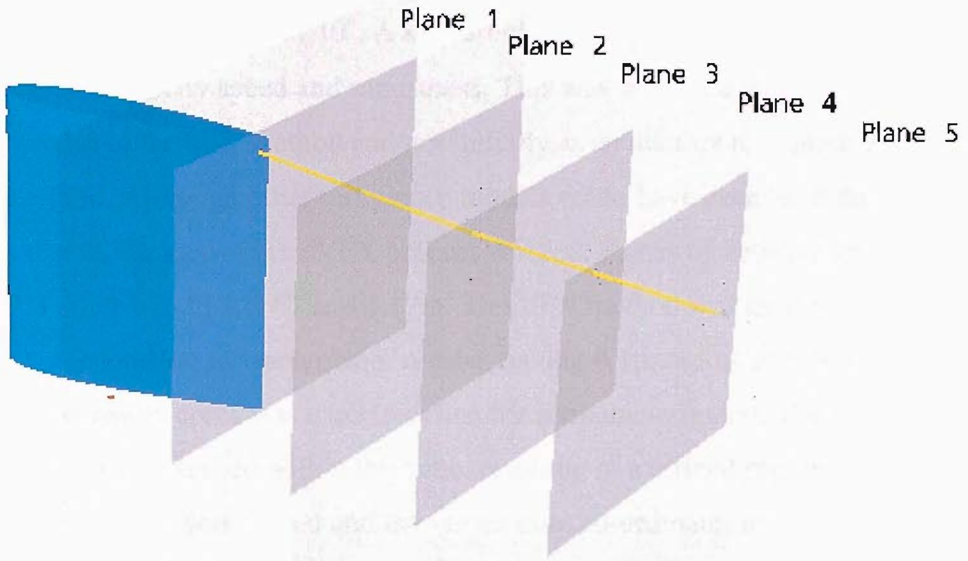


Figure 4.1 - Planes where VFX is performed for a wing

4.1 Wing test case

To test the VFX method it was decided to validate it with an experimental case. A NACA0020 wing operating at 10^{O} angle of attack was chosen. The wing was tested in the 11' x8' George Mitchell [101] low speed wind tunnel at the University of Southampton and the recorded data included measurements of wing surface pressures. The wing has a 1.0m span and 0.667m chord.

The numerical model was an approximation of the wind tunnel experiment, neglecting the blockage effects of the walls. The inlet boundary was located at x= –3.0c and the outlet at x= 5.2c. The side walls were located at y= ±3.0c and the roof at z= 4.5c. The wing leading edge root was at the origin. The floor is defined as a symmetry plane in order to remove the need to capture the groundplane boundary layer. For similar reasons the roof and walls are defined as openings and a velocity defined on those boundaries.

The wing was modelled using an unstructured mesh using only tetrahedrons. The base mesh had 315,566 cells. The cells were clustered near the wing surface, with a maximum edge length of 0.045c. The maximum y^{+} was 100. A cylinder of finer cells also having a maximum edge length of 0.045c extended downstream from the tip aligned with the x axis. The wing was operating in a uniform free stream velocity of 20m/s at an angle of attack of 10°. A k-ε turbulence model with wall functions was used for its simplicity speed and robustness. This was used as a test case for the development of theVFX method and not initially as an attempt to capture accurately the flow field. Although other turbulence models could have been used the aim was to ensure that all the steps in the VFX process worked. Planes of velocity were extracted at 0.67, 1.17, 1.67, 2.17, 2.67 and 3.17m. The VFX method was used to compute the vortex cores location for each plane. A tube, having 0.1m radius and its axis passing through the vortex cores was used to define the refinement region. The maximum cell edge length was specified within this tube, resulting in a refined region in the mesh. The resulting mesh was solved and the vortex core co-ordinates updated. Three iterations were carried out; each time the maximum edge length within the tube was decreased. The refined meshes had 518,997, 819,289 and 1,797,200 cells respectively.

Mesh	Cells	Nodes	Max. edge length in tube	Points across vortex core
Base	315,566	56,461	0.045c	8
Refine 1	518,977	91,397	0.030c	12
Refine 2	819,289	142,870	0.022c	15
Refine 3	1,767,200	305,113	0.015c	23

Table 4-1 - Grids for wing test case

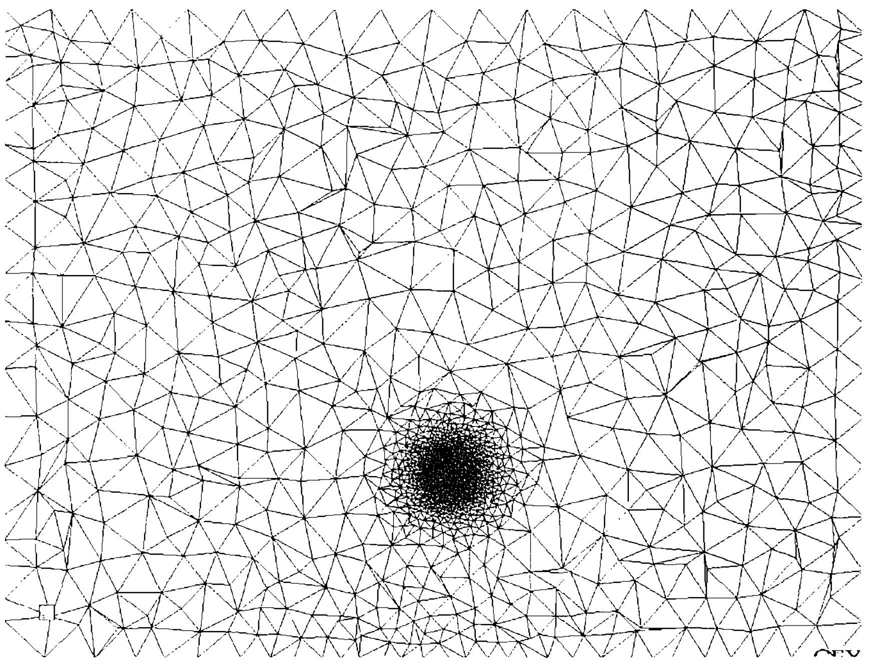


Figure 4.2 - Refined mesh 3. Plane 1.17m

From the results it can be seen that as the number of cells in the vicinity of the vortex core increases the vortex position changes. For the two finer meshes the location of the vortex core line is fairly constant. The vortex shedding off the wing is not located at the tip but further in as expected from theory and experimental observations. The vortex contracts to 80% of the span as it moves 4 chord lengths downstream. This positional dependency of the vortex on grid spacing means that methods like Spall [43], which use a grid with points clustered *a priori*, run into difficulties even for such a relatively simple case.

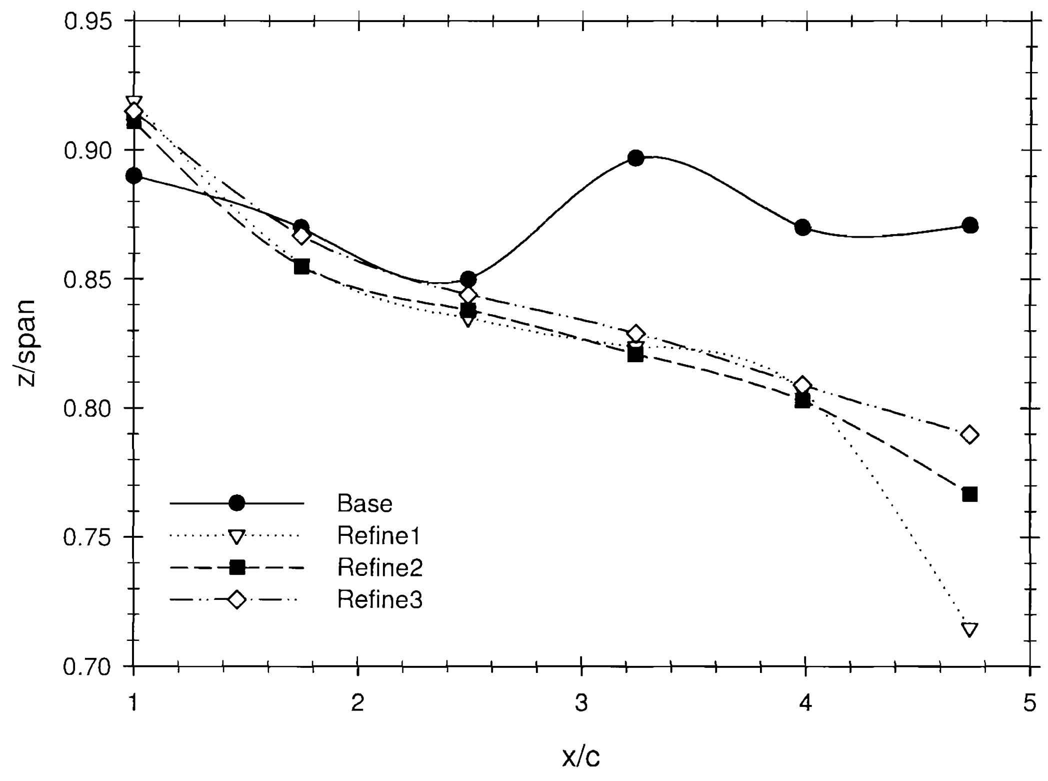


Figure 4.3 Spanwise position of vortex with different grids

Inspecting the vortex velocities at 0.75c downstream of the trailing edge for the different meshes it can be seen that the results are similar to Dacles and Zilliac [41]. From Figure 4.4 it can be seen that the vortex velocities change significantly with grid spacing. As the grid resolution increases the vortex core radius decreases. The two finer grid spacings give similar results, implying that at least 15 points are needed in the vortex core to capture the flow accurately. This agrees well with Dacles and Zilliac who recommend 18 points through the vortex core.

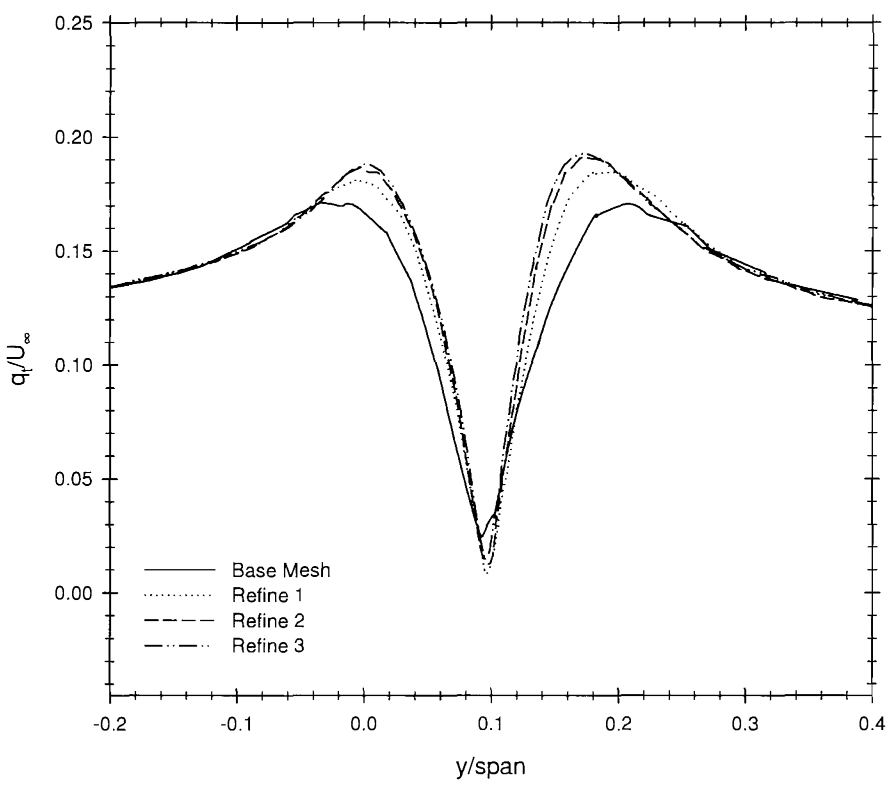


Figure 4.4 – Vortex velocities for the different meshes

Comparing the surface pressures calculated using the finest mesh with experimental data [101] good agreement can be seen (Figure 4.5). However, the results do not show the large pressure drop towards the trailing edge. This is because the clustering is not extended upstream of the trailing edge, and the influence of the vortex on the surface pressures will not be fully captured.

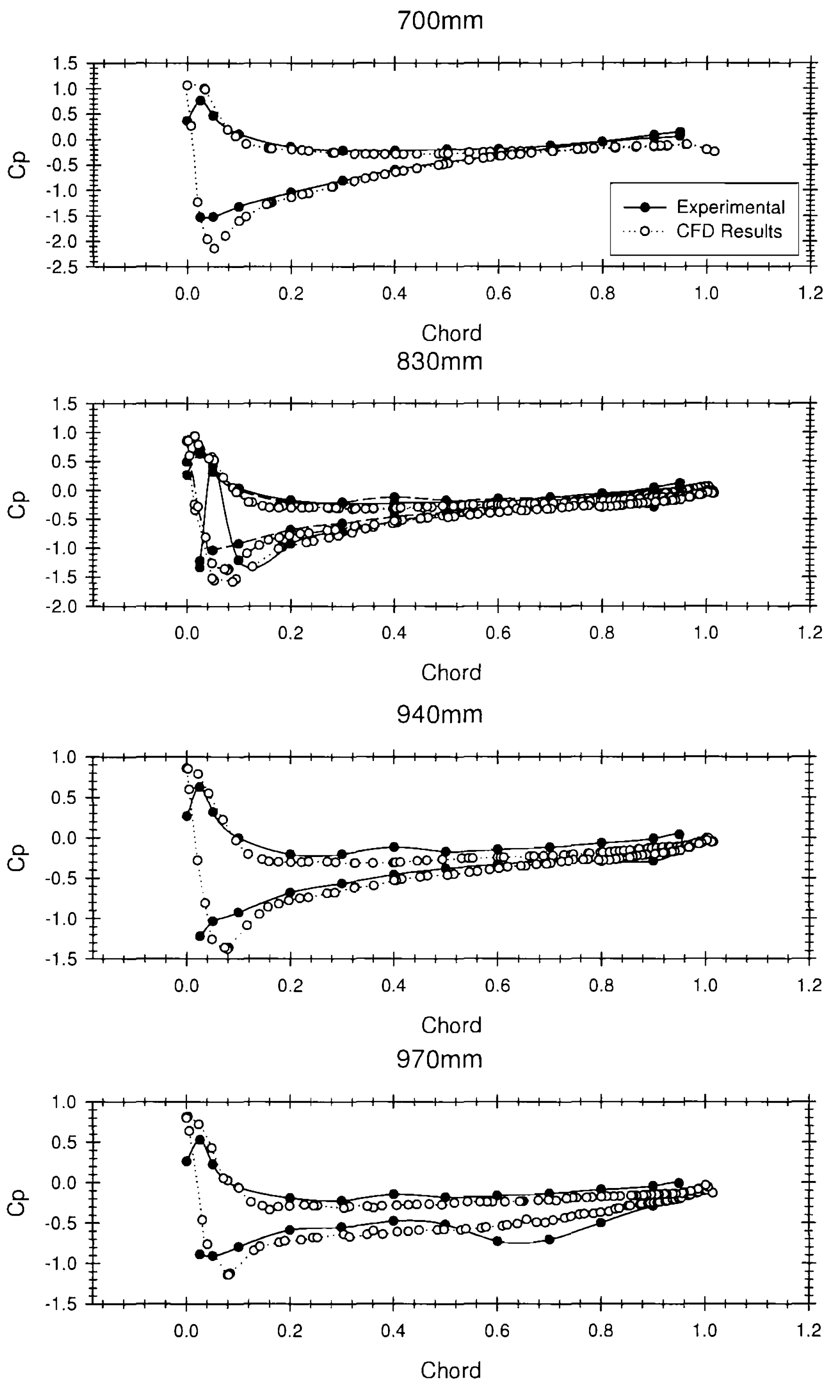


Figure 4.5 – C_P comparison for NACA0020 wing

4.2 Initial grid dependency

In order to ensure that the method described previously is independent of the initial grid, the same geometry was modelled using a different meshing strategy. A different meshing tool was used, ***ICEM 4 CFD*** [102], to generate the mesh. The base mesh compromised of tetrahedra with a different mesh density and distribution than the previous base mesh. The mesh was clustered in the vicinity of a line extending downstream from the wing tip at angle 5° from the x axis, even though the angle of attack was 10°. This is to prove that the initial clustering position is not crucial to the final solution. In fact no clustering is necessary for the initial mesh as long as the initial grid density is fine enough to resolve some of the tip vortex. However since a very coarse base mesh is used a refined region was included.

The same iterative strategy as described above was used, updating the position of the grid refinement and at the same time reducing the grid spacing. After three refinements the position of the vortex was within the convergence criterion of the first solution. This proves that the method is independent of the initial grid.

4.3 Comparison with experimental data

For better validation of the procedure detailed wake measurements were deemed necessary in order to study the evolution of the tip vortex and wake. A numerical model of a wing was compared to experimental data from a wind tunnel model. The experiments were performed in a 0.9m x 0.6m, open circuit wind tunnel, operated at a flow speed of 19.0 m/s. The wing had a chord of 0.45m and a geometric aspect ratio of 1.0. A NACA0012 section was used with constant thickness to chord ratio along the wing [103]. Particle Image Velocimetry (PIV) and pitot tube data were available for the steady case.

The numerical model consisted of a tetrahedral mesh with prisms layers on the wing and tunnel walls to capture the viscous sub layer. The prism layer had a 10mm height on the wing (y^{+} of 50) and 30mm on the tunnel walls. The VFX procedure was implemented, using 5 planes downstream of the wing and the mesh refined on the vortex core line. The position of the vortex core was settled after a few cycles as discussed previously.

After the vortex line had settled the number of refined layers grown from the vortex core line was varied. The cell edge was increased to keep the mesh size constant. The mesh size can rise quickly with increasing layers grown from the vortex core line. This is because the volume of the refined mesh increases with respect to distance squared from the vortex core. For example 10 layers will have 4 times the cells as 5 layers grown from the vortex core line.

From the results it can be seen that there is a trade off between cell size and the number of layers grown. If the refined mesh does not extent far enough away from the vortex core then the vortex is not captured correctly. As the number of layers of refined cells grown from the vortex centre increases we can see that the vortex core radius decreases and the maximum circumferential velocity increases (Figure 4.6). However, since the cell size is increased with increasing number of layers to keep the mesh size the same there comes a point where the mesh is not fine enough in the vortex core to refine the vortex correctly.

From Figure 4.6 we can see that the vortex core radius is over predicted by 1-2% of chord. The circumferential velocity is under predicted by 20%. However these are the mean circumferential velocities around circles with their centres at the vortex core. Any error is thus cumulative. In addition the PIV data is an average of multiple images over time. Comparing the velocities through the vortex along the x and y direction (Figure 4.7) we can clearly see where the discrepancy occurs.

The velocities for the y direction (perpendicular to the wing) are in very good agreement with the PIV data. The core radius is 11% of chord from the CFD simulation and 10% from the PIV data. The maximum velocities are also in good agreement.

For the spanwise direction it is different. The predicted maximum velocities are well below the PIV data. The outermost position of maximum velocity is within 2% of chord, however in the wake the results are worse. This might be due to the influence of the wake on the tip vortex. The mesh in the wake region is quite coarse and not captured correctly.

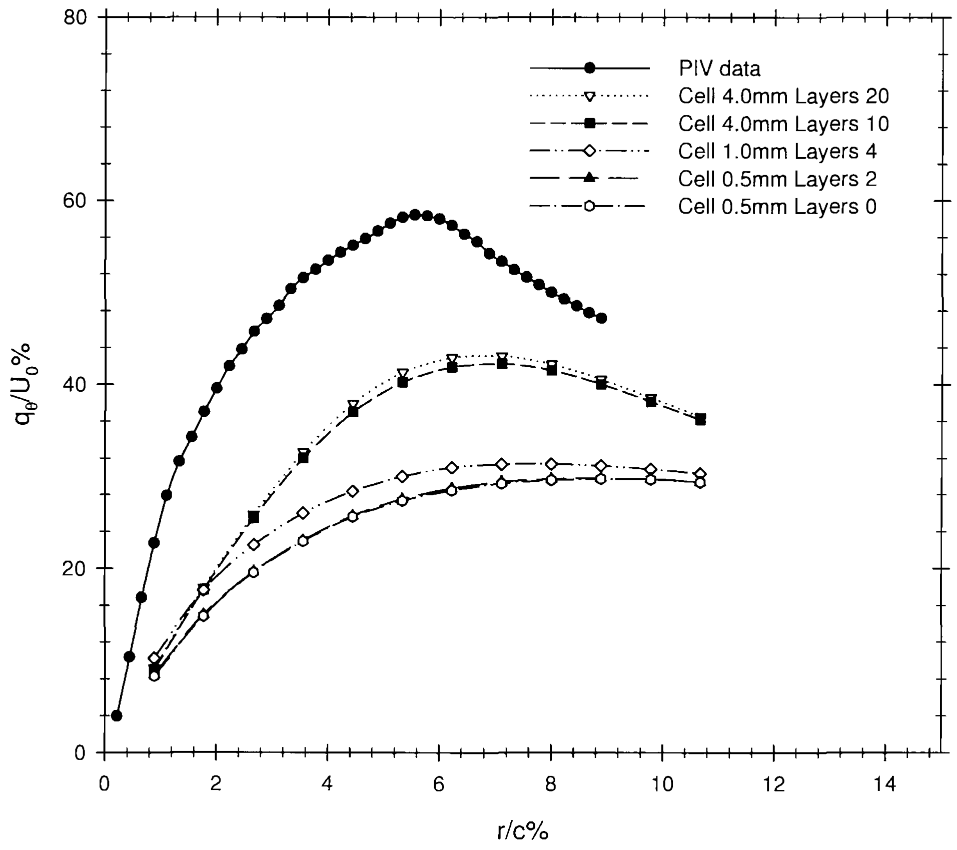


Figure 4.6 – Average circumferential velocity from vortex core

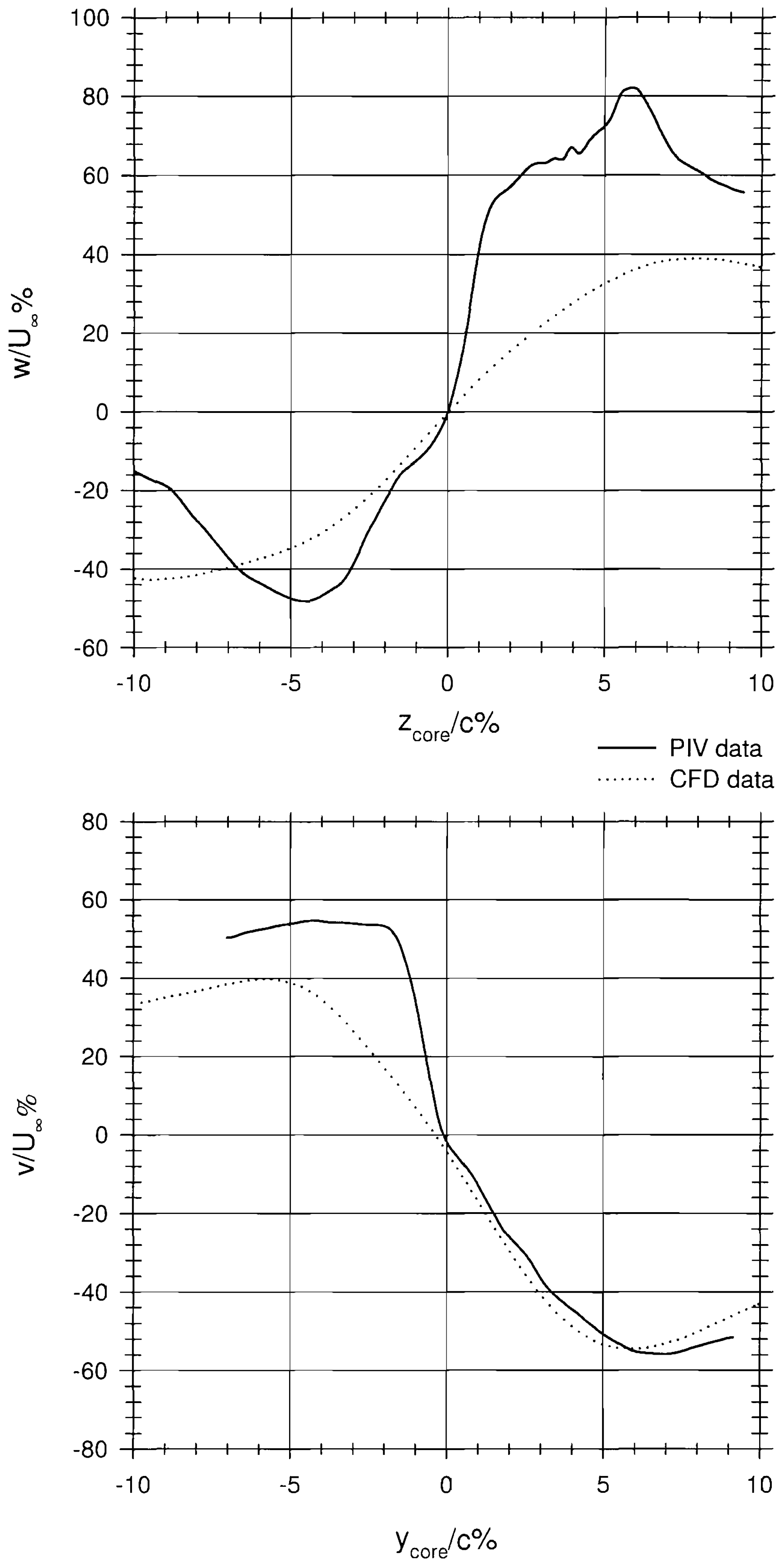


Figure 4.7 – Velocities through the tip vortex. *Top*: Spanwise direction (z). *Bottom:* Parallel to the tunnel floor (y)

The forces on the lifting surface vary significantly with grid refinement in the vortex region. It can be seen that for the very fine mesh, which is only concentrated at the vortex core, the lift and drag are out by 9.3% and 2.8% respectively. As the region of the refined mesh grows out from the vortex radius the results tend closer to the experimental data. The change in lift is very small but the drag improves significantly. The lifting surface is operating near its stall angle and small changes in the angle of attack result in large changes in the forces. Also the flow is unsteady in real life and the lifting surface probably transitions between a partially separated and fully attached condition.

	Lift (N)		Drag (N)		C_L	C_D
	Pressure	Viscous	Pressure	Viscous		
Cell 0.5mm Layers 0	25.72	-0.036	2.665	0.482	0.59	0.073
Cell 0.5mm Layers 2	25.703	-0.0356	2.665	0.481	0.59	0.073
Cell 1.0mm Layers 4	25.892	-0.0359	2.678	0.482	0.6	0.073
Cell 4.0mm Layers 10	25.395	-0.0387	2.605	0.484	0.59	0.071
Cell 4.0mm Layers 20	25.528	-0.0404	2.610	0.480	0.59	0.071
Experiment	-	-	-	-	0.48	-

Table 4-2 - Comparison of lift and drag for different vortex refinement.

4.4 Summary

The Vortfind method has been extended successfully to three dimensional flows. The resulting VFX method has been tested on a wing and was able to track the tip vortex downstream of the foil. The mesh was refined in the region identified using the VFX identified vortex core. The vortex core position stabilised with each progressive refinement and the vortex propagated further downstream.

Detailed wake comparisons have been made for a second wing tested in a wind tunnel where PIV data was available. Although the results have shown an improvement with the refined mesh as far as the tip vortex definition is concerned, there were discrepancies in the forces. It is believed that these were due to the insufficient resolution of the boundary layer wake which is an important flow feature. Resolving the boundary layer shear layer far downstream represents a difficult task, especially for marine propellers where the shear layer is of a helicoidal form. Work by Stanier [30] shows that resolving the boundary layer wake significantly improves the results and Sanchez-Caja *et al.* [104] demonstrate that the wake structure deteriorates very quickly outside the fine mesh region. Thus it would be beneficial to develop a wake identification algorithm that can be used in conjunction with VFX.

5 Adaptive Wake capture

Downstream of any body exists a region of slower moving fluid known as a wake. For a lifting surface, operating at small angles of attack, this is a thin shear layer region formed when the fluid in the boundary layers from the upper and lower surfaces merge at the trailing edge and then extend downstream. To capture this thin shear layer a refined mesh is required. A method for identifying the position of the wake using simple techniques is investigated and described in this chapter.

5.1 Identifying the wake

In order to follow the same philosophy with the VFX method the procedure for identifying the wake must be based on simple mathematics. Ideally, it must not use derivatives or complex functions of the solution.

The wake consists of a small region behind the aerofoil with a velocity deficit. If a threshold velocity deficit is selected the regions in the flow having lower velocity can be identified. The velocity deficit regions are identified from the same plane data used for the VFX method. For the subsequent calculations the shear layers due to the tunnel walls are ignored otherwise they will be erroneously selected as well. A region 0.05m from any wall was therefore masked out. This leaves the points belonging to the wake but also the tip vortex region. The velocity deficit region due to the vortex extends outwards of the geometric span of the aerofoil. If all this points are used to determine the wake it causes problems. The wake behind an aerofoil only extends to the vortex core. The position of the vortex core is already calculated using VFX. Any points outwards of the vortex core can also be masked out.

A curve is fitted to the remaining points using a moving average filter. This is performed for all the planes downstream. A refined grid can then be specified in the region of the wake. To reduce the number of cells required in the wake a similar approach to that used for shear layers near walls can be used. By using high aspect prisms aligned to the wake the grid spacing along the wake can be kept large whereas the transverse grid spacing can be reduced to capture the shear layer.

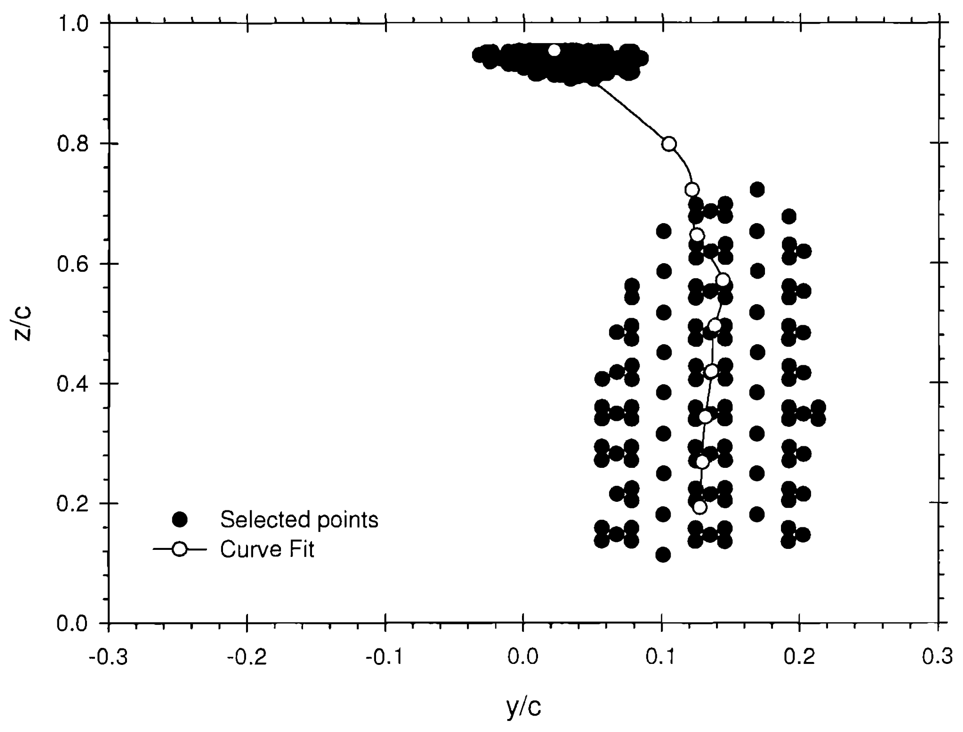


Figure 5.1 – Selected points having velocity deficit after masking

The wake capture method worked well within one chord length of the trailing edge (Figure 5.1). Downstream the velocity deficit in the coarse wake was less and thus the wake region was not selected. Only the vortex deficit region had a low enough value to be selected (Figure 5.2). One possible solution was to select different velocity thresholds for each plane, with the threshold velocity increasing away from the aerofoil. This was not deemed a feasible solution since an appropriate threshold velocity is not known *a priori*. Such a method would require a time consuming trial and error approach.

A more generic and robust solution was developed. The spanwise distance between the root of the foil and the vortex core is subdivided into a number of strips. The areas near the tunnel walls and floor area masked out, as explained above, and the remaining points for each strip are sorted in ascending velocity magnitude. The top 5% of the points are selected for each strip. From thereon the same approach is used as described previously. This method ensures that the entire wake from the root to the tip vortex is selected and all the way downstream (Figure 5.3). Also it is more robust and less sensitive to user input.

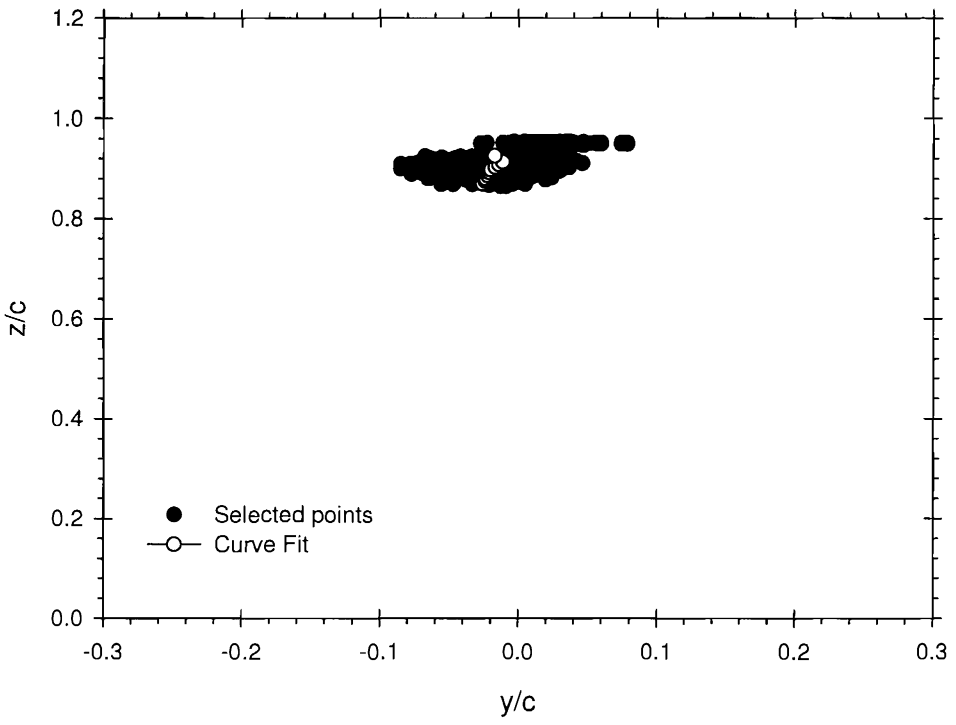


Figure 5.2 – Selected points having velocity deficit at a plane one chord length downstream of the trailing edge

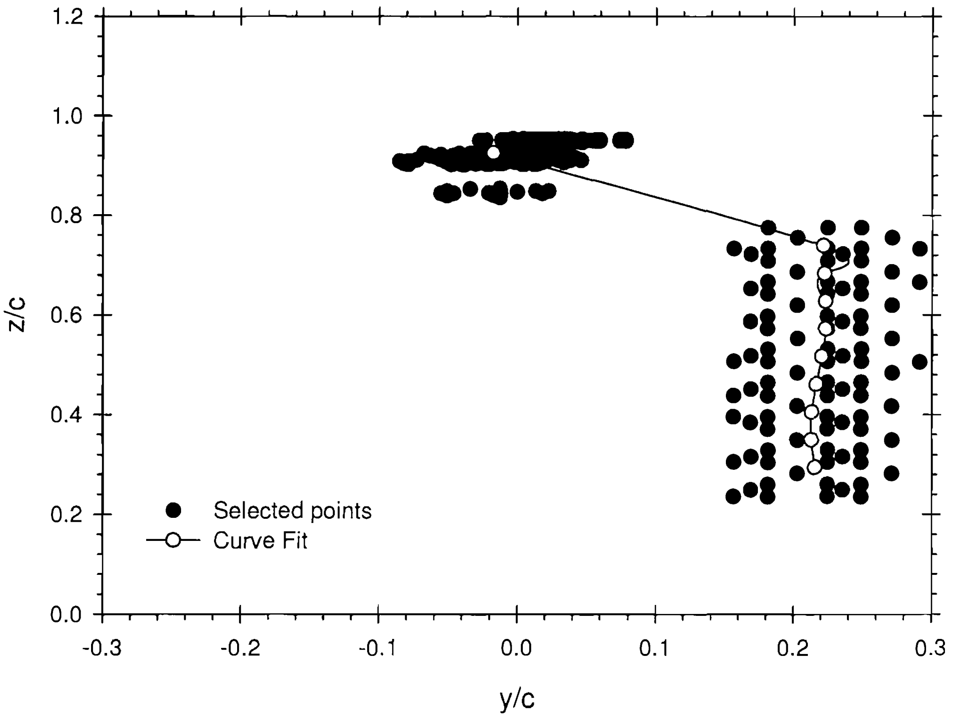


Figure 5.3 – Selected points using strips and selecting top points at one chord length downstream of trailing edge

5.2 *Wake mesh*

A surface was lofted through the wake lines identified from the wake algorithm and the trailing edge of the wing. The resulting wake shape is as expected from classic aerodynamic theory. It extends downstream of the wing and wraps up around the tip vortex (Figure 5.4). The same mesh was used as for the standard models without the wake capture. This consisted of a fine mesh on the wing surface and in the vortex region. In addition a ten cell inflation layer on the wing and wall tunnel walls with 1cm and 3cm overall thickness respectively was used.

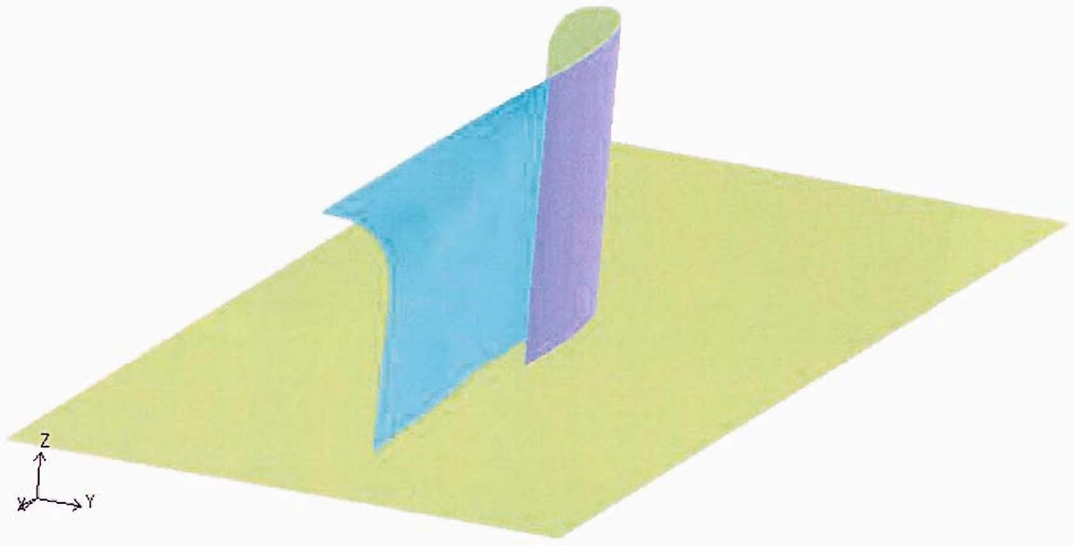


Figure 5.4 – Wake shape captured by the wake identification algorithm

For the wake capture ten layers of prisms were extruded normal to the wake surface captured either side .The prisms matched the inflation layer on the wing. The thickness of the prism layer in the way of the wake was 60mm. The streamwise mesh spacing in the region of the wake is large to reduce the overall size of the mesh. However, the transverse grid spacing is small to refine the shear layer in the wake (Figure 5.5). If the prisms were replaced with tetrahedra having an aspect ratio of one then thirty times more cells would be required to have the same transverse grid spacing.

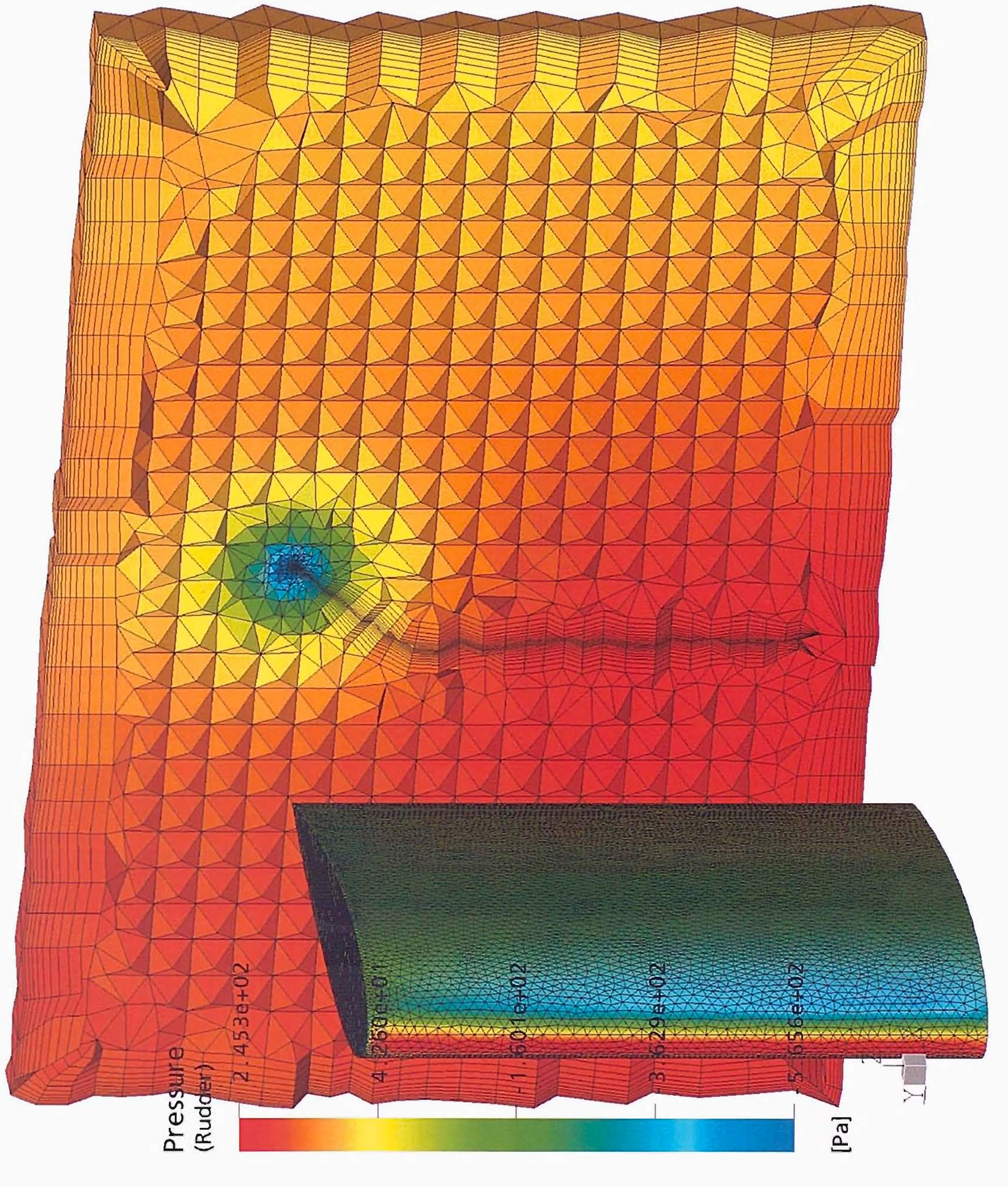
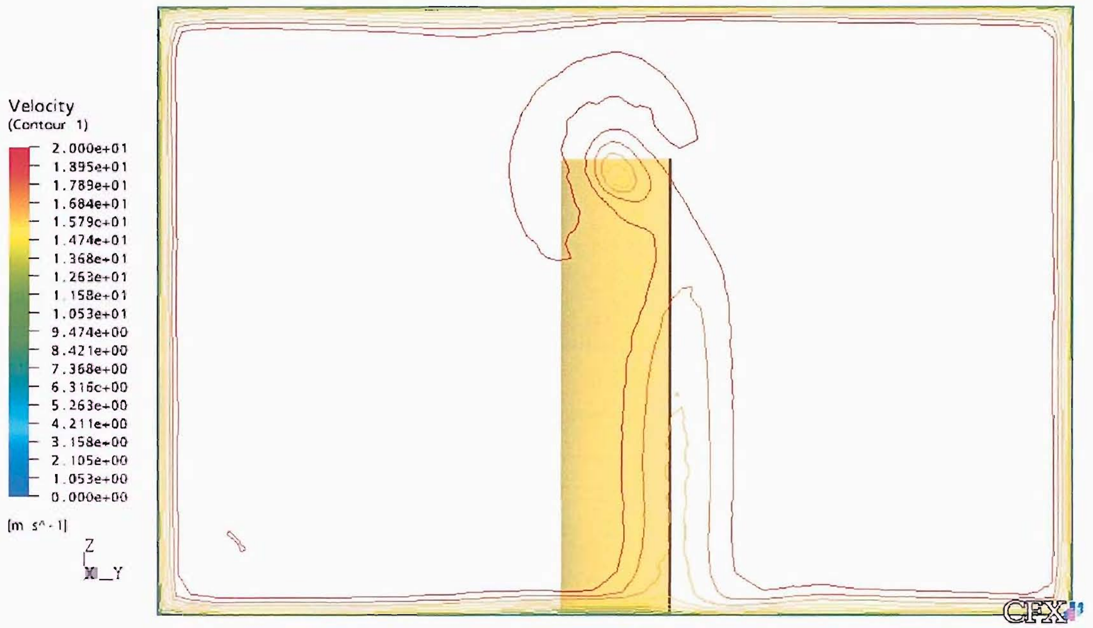


Figure 5.5 – Mesh using wake capture

5.3 Results

Comparing the results for the standard mesh with the prism wake mesh we can see that there is a significant improvement. The minimum velocity for a plane 44.4%of chord downstream of the trailing edge is below 1.57m/s for the wake prism mesh whereas for the standard mesh the velocity is not less than 1.68m/s. In addition the wake prism mesh influences the tip vortex. The velocity deficit area above the vortex is less pronounced with the wake prism mesh.



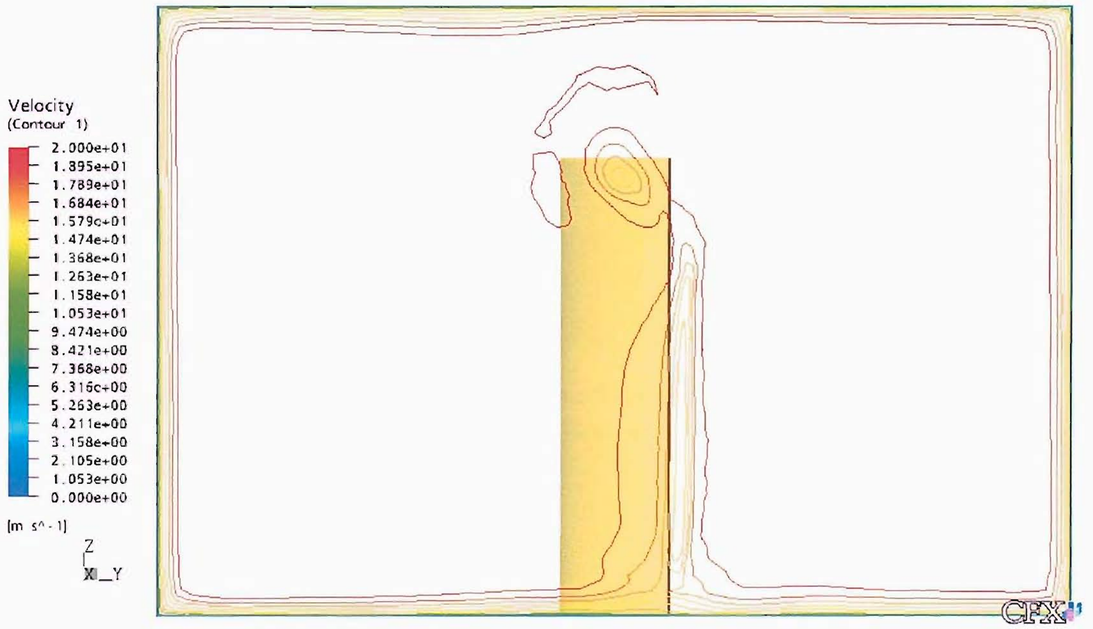
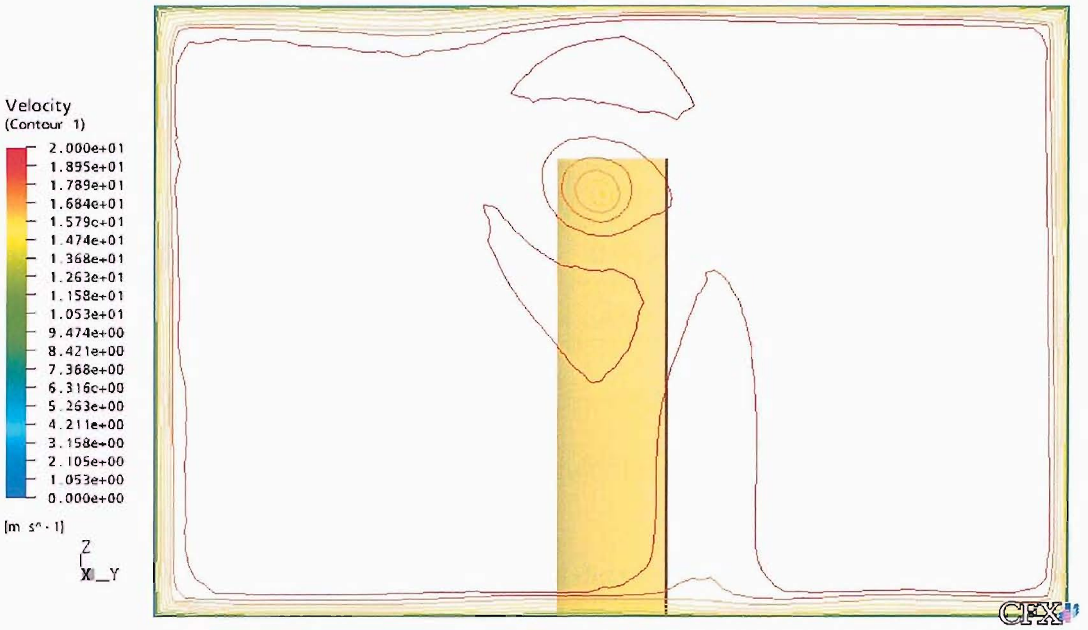


Figure 5.6 – Velocity contours 44.4% of chord downstream of the trailing edge for standard mesh (Top) and prism wake mesh (Bottom)

Similar results are observed for a plane 133.3% of chord downstream of the trailing edge. The same is true for the entire wake downstream of the wing.



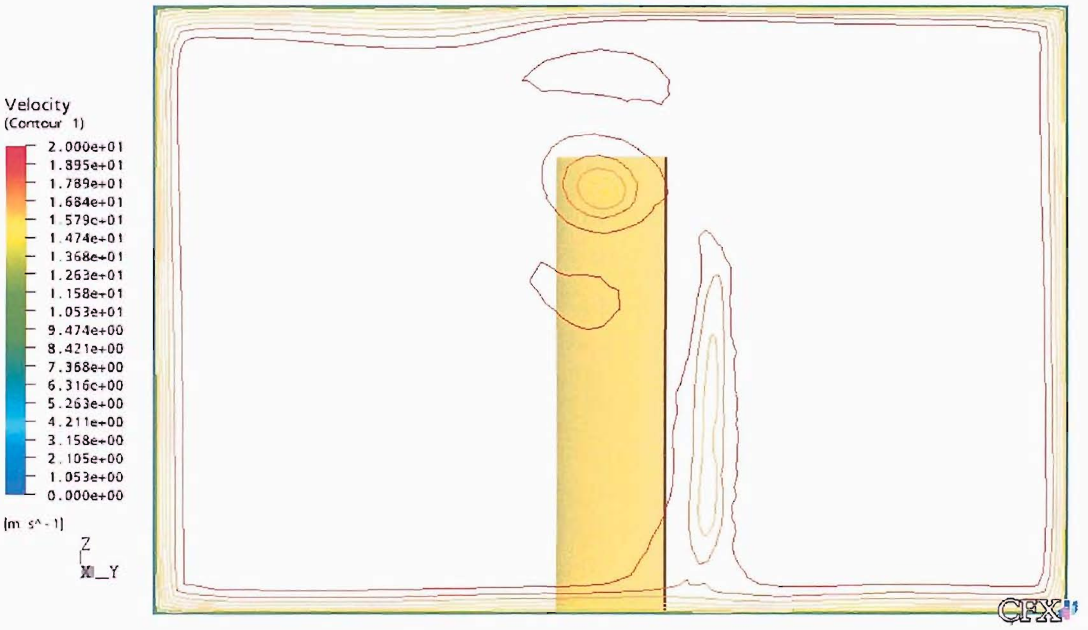


Figure 5.7 – Velocity contours 133.3% of chord downstream of the trailing edge for standard mesh (Top) and prism wake mesh (Bottom)

The change in the forces on the wing is significant. There is a 9.9% reduction in drag and a 5% reduction in lift. The lift to drag ratio increases by 5.3%. There is a reduction in the pressure forces and an increase in the viscous forces (Table 5-1).

	Pressure force (N)			Viscous force (N)			C_L	C_D	L/D
	x	y	z	x	y	z			
Standard	2.665	-25.72	2.712	0.48	0.036	0.015	0.59	0.073	8.17
Prism wake	2.338	-24.40	2.660	0.49	0.045	0.013	0.56	0.065	8.60
Δ%	-12.2	-5.1	-1.9	2.1	24.4	10.3	-5.0	-9.9	5.3
Experiment	-	-	-	-	-	-	0.48	-	7.56

Table 5-1 - Comparison of wake mesh and standard mesh forces

Comparing the results with experimental data from a wake transverse study [103] we can see that the wake is better defined with the prism wake mesh. The results agree more closely with the experimental results. The position of the wake is predicted very well. The velocity deficit is less than the experimental results by about 15% of U/U_0 for the plane immediately downstream of the trailing edge. However for the rest of the downstream planes the velocity deficit in the wake is within 5% of U/U_0 (Figure 5.8).

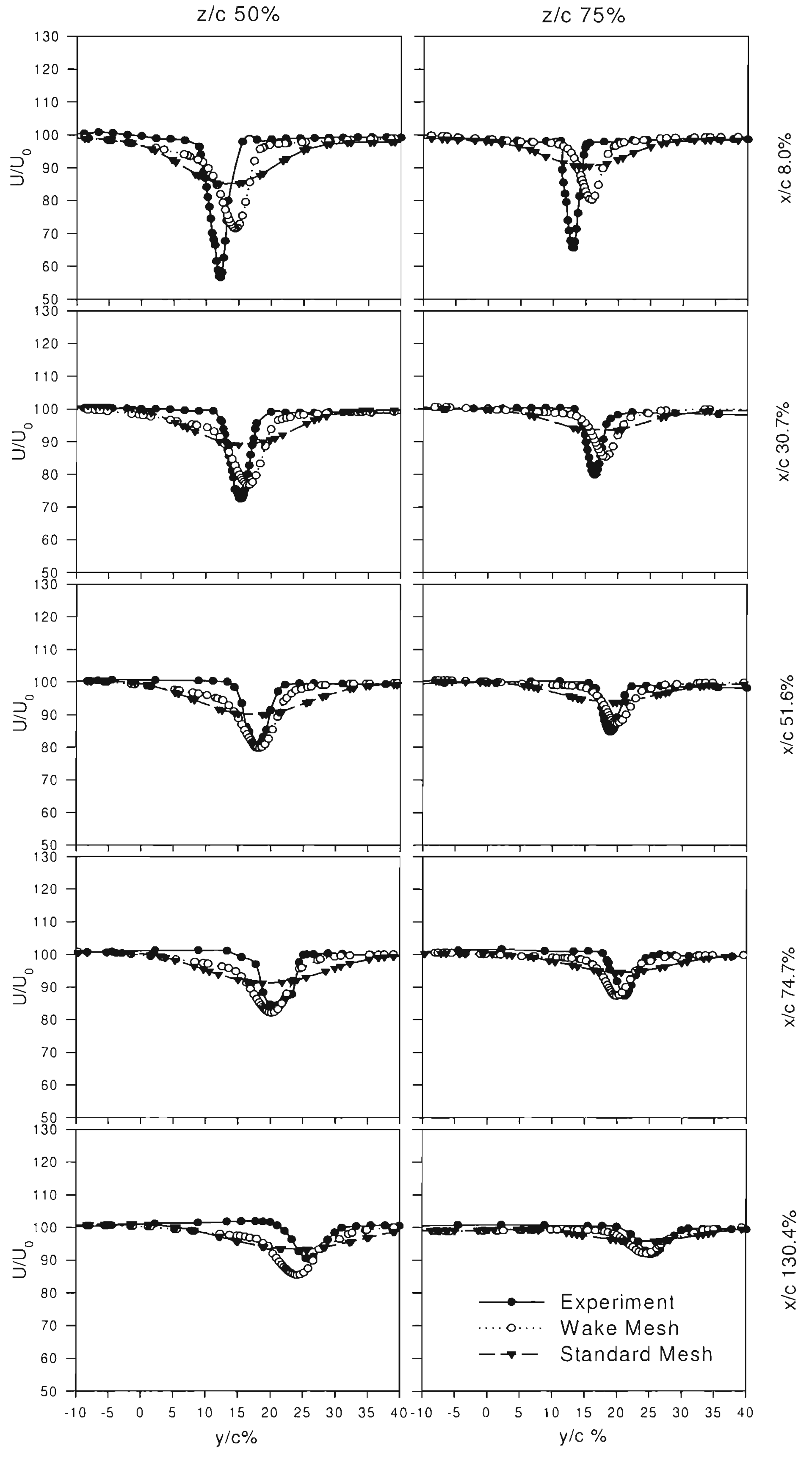


Figure 5.8 – Wake survey comparison

5.4 *Results for Wake and VFX mesh*

A mesh having the same cell size as for the VFX case with 10 layers grown away from the vortex core (see Table 4-2), and having a wake mesh was generated. However no inflation layer was used on the tunnel walls to reduce the mesh size. The effect of the inflation layer having been found to be negligible.

	Lift (N)		Drag (N)		C_L	C_D	L/D
	Pressure	Viscous	Pressure	Viscous			
VFX	25.40	-0.0387	2.605	0.484	0.59	0.071	8.31
Prism wake	24.40	-0.045	2.338	0.49	0.56	0.065	8.62
VFX & Prism wake	21.63	-0.0246	2.556	0.434	0.48	0.068	7.05
Experiment	23.04		3.18		0.48	-	7.61

Table 5-2 - Comparison of forces

From the results we can see that the drag is somewhere between the two meshes. However the is a large reduction in lift. This gives a better lift to drag ratio than the other two meshes which significantly over predict this. Looking at the spanwise loading we can see the typical increase in loading due to the tip vortex.

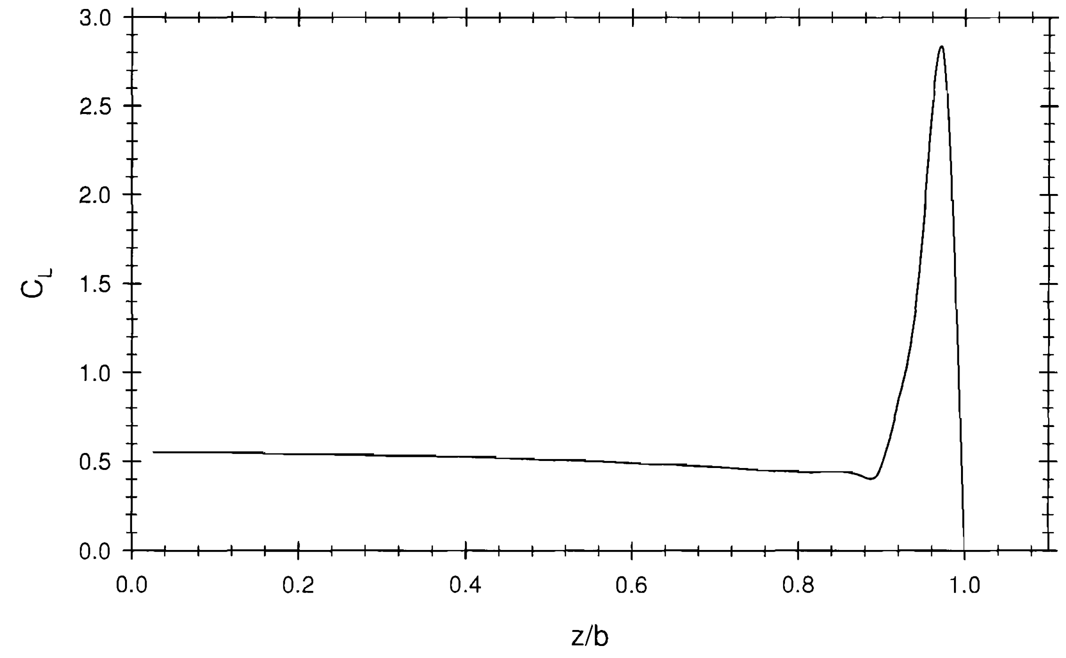


Figure 5.9 – Spanwise loading for NACA0012 wing

5.5 Results for Wake and VFX mesh

The importance of resolving the wake has been shown for a 3-D wing. A simple algorithm for identifying the wake has been developed which can be used in conjunction with the VFX method. The method has been automated and applied to a 3-D wing and the results compared to experimental data. The forces on the wing changed significantly with the wake refinement. In addition, the velocity profiles in the wake agree very well with pitot tube measurements from the wind tunnels tests.

When used in conjunction with the VFX method to resolve the tip vortex as well, the lift changed significantly and coincided with the value obtained from the experiments.

6 Propeller mesh generation

Experience shows that the choice of the grid for a given propeller can influence the convergence of the solution and numerical prediction [30, 105]. The grid density is a crucial factor in capturing the flow features in the fluid. Grid quality is also of importance and must be addressed as reported by the ITTC [106] & Stanier [107] and discussed for hulls by Bull [31]. Mesh generation is a function of the experience and ingenuity of the person involved, the meshing tools available and restrictions imposed by the limitations of the solver.

6.1 Propeller Geometry

In order to define the propeller blade geometry the following system is used [108, 8]. The blade is formed starting with a midchord line defined by the radial distribution of skew angle $\theta_m(r)$ and rake $x_m(r)$. By advancing a distance $\pm\frac{1}{2}\,c(r)$ along a helix of pitch angle $\phi_p(r)$, the blade leading edge and trailing edge are obtained. The surface formed by the helical lines is use as the reference upon which the sections can be built. These sections are defined in standard aerofoil terms by a chordwise distribution of camber $f(s)$ and thickness $t(s)$, where s is a curvilinear coordinate along the helix. A brief description of the transformations developed and used is given below. For a more detailed description of the process see [109].

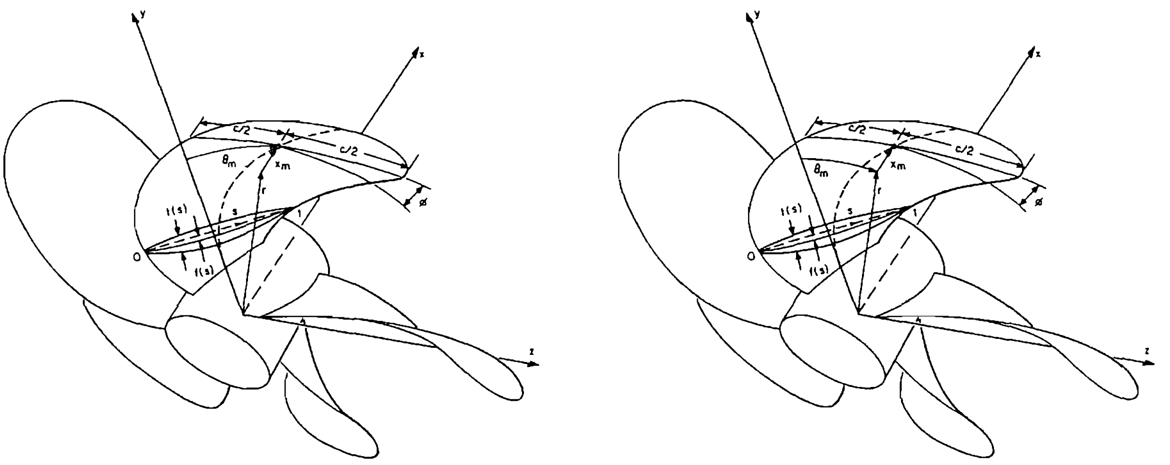


Figure 6.1 - Propeller geometry definition used (Left); Kerwin definition [108] (Right)

The method used differs from the method described in Kerwin [108]. The skew of the propeller is treated differently. The section is skewed along the generator helix for that section whereas in Kerwin's system the section is skewed about the propeller axis. The latter method changes the shape of the blade with varying pitch, whereas the former preserves the same blade shape for all pitches. This makes propeller design easier since the blade shape is not coupled to the pitch. For propellers with no skew such as the DTMB P4119 the two methods are identical. For heavily skewed propellers there is significant variation between the two methods.

Care must be taken that the transformed 2-D sections lie on a cylindrical surface instead of being offset perpendicular to the helix generator. Kouh & Liang [110] and Kouh & Chen [111] neglected to ensure this which can lead to minor geometrical difference which can be significant. Later Kouh *et al.* [111] identified this problem and corrected their methodology.

6.2 Propgen

To generate a mesh for a given propeller geometry a quick and simple tool was required for a wide range of propellers. ***Propgen*** was developed specifically for this purpose. It can handle most propeller geometries, including ducts. It can also generate an inner duct ring for use with tip driven propellers [109]. The generated model is a segment containing one blade only. The model is then assumed to be rotationally symmetric for a steady state case and periodic boundary conditions can be used to model the complete problem. This effectively reduces the grid size reducing memory and computational costs. Complete geometries can also be generated by copying the generated model.

The generation of the propeller requires an input of standard propeller table data and section offsets. The propeller geometry is constructed from a set of section curves. These sections ca be generated for any given radius by defining the chord, thickness, skew, rake, pitch and 2-D section shape. The 2-D section is mapped onto a cylindrical surface according to the specified variables using a transformation matrix (Eq(1.21)). For a more details on the section mapping see [109].

$$\phi = \tan^{-1}\left(\frac{P}{2\pi r}\right)$$

$$\psi = \frac{x_s}{\sqrt{r^2 + (P/2\pi)^2}}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} r\cos\left(\psi - \frac{y_s}{r}\sin\phi\right) \\ \frac{P\psi}{2\pi} + y_s\cos\phi \\ -r\sin\left(\psi - \frac{y_s}{r}\sin\phi\right) \end{bmatrix} \tag{1.21}$$

For use with panel codes an automatic wake sheet can be generated. The shape of the wake sheet depends on the section characteristics and advance ratio. Contraction effects can also be included. Any of the automatic variables can be manually specified to provide more control and flexibility. For more information of the wake model see [109].

Once the section curves have been generated they can be exported to another program which can loft a surface through the sections to generate the blade. There are several supported file outputs supported by ***Propgen***: a fleximesh file for use with ***Adaptflexi*** [112], or script files for ***Gridgen***, ***CFX Build*** and ***ICEM***. An additional version of ***Propgen*** called ***Solidprop*** can work with ***Solidworks*** to provide an iges file which can be imported into many Computer Aided Design packages. ***Propgen*** has been used successfully to carry out the hydrodynamic optimisation of an electric tip driven thruster using a panel code. In addition it provided the propeller geometry of a Wageninnen propeller for CNC machining.

For use with the commercial grid generation package ***Gridgen***, ***Propgen*** generates a script file called *glyph*. ***Gridgen*** supports both structured and unstructured meshes and can export grids in most commercial file formats. The script file contains the geometry of the sections that define the propeller as well as connecting information. The mesh parameters and controls are also contained in the script file such that the whole process can be automated.

Gridgen uses transfinite interpolation [113] for faces constructed from their outer edges. For complex curvatures this results in the surface being misinterpreted [109]. This can be resolved by splitting the surface into smaller surfaces but this restricts the mesh. An alternate method is to define the geometry using an *iges* file containing surfaces instead of curves and project the mesh onto that surface.

To generate the surfaces the solid modelling package ***Solidworks*** was used to create the propeller and domain surfaces for use with ***Gridgen*** or any other software capable of importing *iges* files. The propeller geometry is automatically generated in the solid modelling package using a built in program ***Solidprop***, written for this purpose.

Gridgen is a powerful structured grid generator providing excellent control on the mesh. However, the unstructured capabilities of ***Gridgen*** were found to be limited and the mesh generation process does not exploit many of the benefits of unstructured meshes. ***ICEM*** was found to be more suited for unstructured meshes and has excellent features. In addition ***ICEM*** has the capability of automatically creating inflation layers which are crucial in obtaining good results when using unstructured meshes as demonstrated for the wing. A script file for ***ICEM*** can be exported from ***Propgen*** to automatically generate marine propeller meshes. In addition the propeller surface can be created within ***ICEM*** from the existing curves producing a surface. The blade was created from multiple B-spline surfaces each created from four surrounding curves. The four curves were the two half sections and the leading and trailing edge curves between those two sections. Wrapping the surface around the blade using only the two complete sections was found to be inadequate. The leading and trailing edge curvature was not reproduced correctly. The same was found if one surface was lofted though all the sections.

6.3 Mesh considerations

Generating structured meshes for marine propellers can be very time consuming. To build a structured mesh topology can be very complex resulting in many blocks. The topology can be simplified by using degenerate blocks where a block has 5 faces instead of six, but several solvers do not have this capability. The alternative and most popular way is to truncate the blade tip such that there is a finite chord at the blade tip [30]. In reality most open water marine propellers have zero chord at the blade tip. This gives acceptable predictions as far as K_T and K_Q are concerned. However, the tip vortex is strongly dependent on the tip geometry and thus must be modelled as accurately as possible. Unstructured meshes are more versatile when modelling complex geometries and do not encounter this problem. In addition they are quicker to generate only requiring the bounding faces of the volume to generate the mesh.

A structured mesh was generated using ***Propgen*** and ***Gridgen***. The blade tip was truncated to simplify the blocking structure. However it was decided quite early on that this approximation was not acceptable for this research and instead of using a more complicated blocking structure it was decided to use unstructured meshes instead. This decision was also justified by the adaptive mesh friendliness of unstructured meshes, which would prove advantageous at a later stage.

7 VFX procedure for Propeller modelling

In this chapter a scheme is presented for the application of the VFX method to a marine propeller. The algorithm is described step by step and the effects of different parameters discussed. The developed algorithm is later applied to two marine propellers in chapters 8 & 9 and the results compared to available experimental data.

The tip vortex structure for a marine propeller is more complicated than the tip vortex of a wing. The vortex core line follows a helical like path downstream of the blade tip with varying pitch and contraction. In order to capture this complex vortex an automated algorithm was devised for use with VFX. The only external input required from the user is the desired plane spacing.

The blade tip is taken as the first jump point for the algorithm. A circular plane having a radius 0.25 of the propeller diameter centred about the first jump point is used to extract the stationary frame velocities. The velocities are extracted at the points where the cell edges intersect with the plane. The VFX method is used to locate the vortex core on the first plane. Using the velocity at the predicted vortex core a new jump point is projected a given distance downstream. The rotation of the domain must be taken into account when projecting the new jump point.

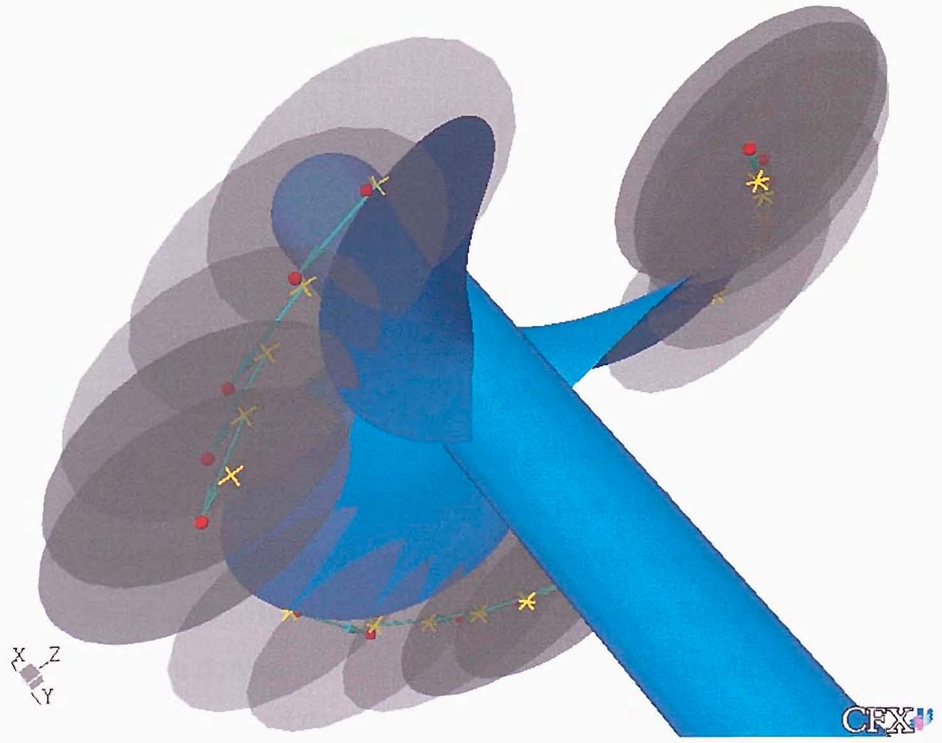


Figure 7.1 – VFX procedure for propellers. Red spheres are the jump points and yellow crosses the VFX vortex cores.

A plane is then used to extract the velocities at the new jump point and the procedure is repeated again. **Figure 7.1** shows a graphical representation of this procedure on the DTMB P4119 propeller. The red spheres represent the jump points and the Yellow crosses the VFX vortex cores.

Using the blade tip as the starting point for the algorithm means that the vortex is not captured upstream. However the above procedure can be performed in the upstream direction as well to track the vortex upstream of the starting point. A smaller plane spacing is recommended for the upstream tracking since the tip vortex is likely to be formed next to the leading edge of the blade which usually has a higher curvature than the vortex helix. Thus a finer spacing will help to capture the vortex more accurately.

The plane spacing and bounds can be varied. Because of the complex nature of the vortex core geometry a small spacing of 50mm was chosen for the planes. If a bigger spacing is used with small plane bounds then there is a risk that the jump point will be too far away from the vortex core and it will not lie in the plane. It is clear that a small plane spacing can be used with small planes; whereas a bigger plane spacing requires larger planes. The computational effort increases both with number of planes and plane size. However the computational effort increases with plane size squared so it is better to have more small planes.

This procedure was tested on the DTMB P4119 using the velocities relative to the rotating mesh but proved unsuccessful. The procedure was similar to the one described above. The first jump point was the blade tip and the next plane was projected in the direction of the relative velocity at the vortex core. The new plane was normal to the velocity vector at the previous vortex core point. This procedure proved highly sensitive to the orientation of the plane and failed after a few iterations.

The vortex decays very quickly outside the refined mesh and thus does not propagate substantially with each mesh refinement, thus a large number of mesh iterations are required. In order to increase the distance the vortex propagates with each iteration, the characteristics of the identified vortex core line were used to define a predicted vortex core downstream of the identified one. The predicted vortex core line is a helix with its pitch and contraction the same as the average pitch and contraction of the identified vortex core line. The mesh was then refined for both identified and predicted vortex core lines. As long as the predicted vortex is near the vortex region the number of refinement iterations is reduced dramatically. It is possible that after the use of the predicted vortex that only the VFX method needs to be applied once to locate the resulting vortex core line if needed.

8 DTMB P4119 and numerical model

The DTMB P4119 is a three bladed open water marine propeller tested at the David Taylor Model Basin [77]. It is a propeller frequently used for numerical method validation and was used at the Propeller RANS/Panel workshop [114]. Extensive tests were performed both in the towing tank and water tunnel and detailed data is available.

The propeller was tested in the 24"circulating water tunnel at the David Taylor Model Basin and Laser Doppler Velocimetry (LDV) data for the boundary layer and wake were obtained. For the open water tests the propeller was tested in a conventional towing tank.

The geometry of the water tunnel is shown in Figure 8.1. The complex geometry of the test section was not modelled in the numerical analysis. The numerical model was a 120° segment containing one blade of the propeller. Rotational image boundaries were used to model the whole propeller. This reduces the computational size of the domain. The outer boundary was a cylinder at a diameter of 38" which is equivalent to 3 times the propeller diameter. The outer boundary was set as a free slip wall. The geometry of the drive shaft was modelled and extended all the way to the outlet boundary. The support struts for the drive shaft were not modelled.

The base mesh for the DTMB P4119 consisted of 315,114 cells. No clustering of cells was performed apart from near the surfaces of the propeller. An inflation layer on the propeller blade was also incorporated consisting of 51,000 prisms giving a y^{+} value ranging from 30 to 40. The mesh was then progressively refined in the region of the tip vortex according to the solution and the VFX method. The maximum cell edge length in the refinement area was progressively decreased for each mesh.

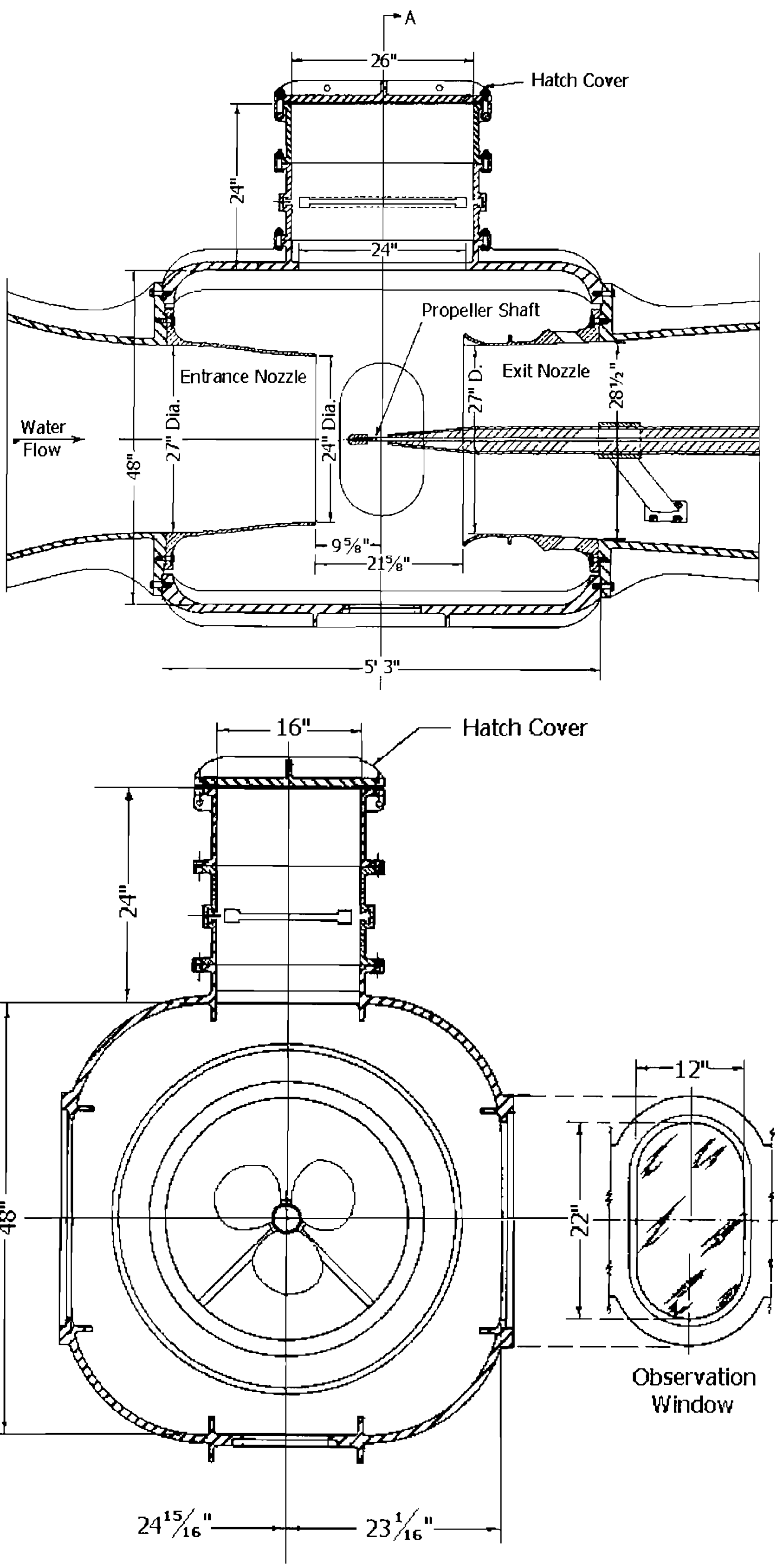


Figure 8.1 – Test section of the 24" VPWT at the David Taylor Model Basin

8.1 Results

After each solution the vortex is identified using VFX and the mesh refined in the vicinity of the vortex core. The resulting meshes are shown in Table 8-1. The mesh is refined around the vortex core line. A maximum cell edge length and number of cell layers away from the vortex core line having this property are specified. This results in a cylindrical region of refined cells with the vortex core line being its axis. This is called the outer refinement region. In addition a smaller cylinder with a finer mesh can also be specified the same way called the inner refinement region. As explained previously a predicted vortex core line is also used for refinement. No inner refinement region was specified for the predicted core for any of the meshes.

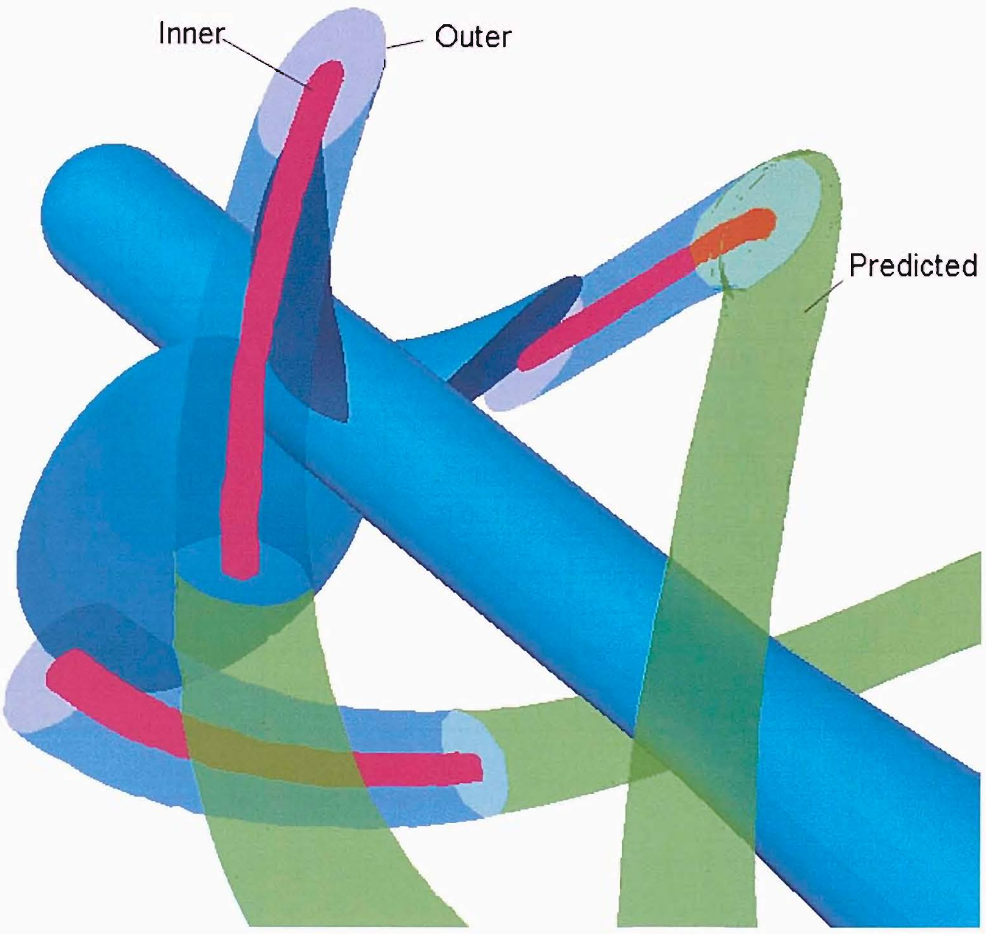


Figure 8.2 – Refinement regions for propeller mesh

As the mesh is refined the vortex propagates further downstream. After 4 refinements the vortex no longer propagated downstream with further refinements. Several mesh densities and strategies were used to propagate the vortex further downstream with no success. A finer mesh was generated near the propeller and the predicted refinement region was shortened in an attempt to cluster more cells near the blade. The predicted region was deemed not to need refinement since the vortex dissipated well upstream. Meshes 5b to5d were the resulting meshes (Table 8-1), however the vortex did not propagate any further than 0.5D.

Use of a transient solution solved the problem and the vortex propagated to 1.3D downstream to the end of the refined mesh where it quickly dissipated within a short distance. This was for mesh 5d using one timestep of 0.1s. This is because the vortex has a tendency to wander even for steady flow problems. Using a steady state scheme smears the wandering vortex over several timesteps where as a transient scheme does not have this problem. By solving the flow as a transient problem no averaging of the flow is performed and thus no smearing of the tip vortex is present. The unsteadiness of the forces is presumed small enough such that there is no need to solve for several timesteps and average the forces.

Mesh	Inner		Outer		Predicted	
	Max length	Layers	Max length	Layers	Max length	Layers
Base	-	-	-	-	-	-
Refined 2	-	-	0.004	5	-	-
Refined 3	-	-	0.003	4	0.005	4
Refined 4	-	-	0.0015	5	0.004	5
Refined 5	-	-	0.001	5	0.003	5
Refined 5b	0.001	1	0.003	4	0.004	5
Refined 5c	0.001	5	0.004	5	0.004	5
Refined 5d	-	-	0.0015	6	0.004	5
Refined 6						

Table 8-1 - Mesh properties for refined meshes

With progressive refinement in the tip vortex region the predicted thrust and torque are in better agreement with the open water experimental results. At the same time the tip vortex is able to propagate further downstream. The over prediction of the torque coefficient is a common problem of RANS predictions and it is accounted to the inability of the k-ε model to predict the stagnation pressure on the blade [115]. The k-ε model has been shown by Bulten *et al.* [115] to produce higher stagnation pressures thus over predicting the torque coefficient.

Mesh	Size	K_T	Δ% K_T exp	K_Q	Δ% K_Q exp
Base	315,114	0.155	6.4	0.0286	2.1
Refined 2	367,247	0.153	4.8	0.0308	9.9
Refined 3	1,380,466	0.152	3.9	0.0305	9.1
Refined 4	1,205,930	0.151	3.6	0.0302	7.8
Refined 5	2,437,495	0.144	-1.2	0.0293	4.6
Refined 5b	1,461,281	0.155	6.5	0.0310	10.7
Refined 5c	1,906,560	0.151	3.6	0.0305	9.1
Refined 5d	2,650,308	0.158	8.2	0.0310	10.7
Refined 6	7,007,050	0.172	17.5	0.0328	17.2
Refined 6 unsteady	7,007,050	0.145	-0.9	0.0292	4.2
Experimental		0.146		0.028	

Table 8-2 - K_T and K_Q variation for DTMB4119 with different meshes

8.2 *Comparison with LDV data*

8.2.1 Section pressure distribution

Comparing the C_P distribution for the section at 0.9 r/R similar results can be seen as compared to the Grenoble workshop [114]. The C_P is over predicted near the leading and trailing edge which was typical of most results. This is because the C_P has been calculated from the pressure on the blade surface whereas the experimental C_P was calculated using the LDV data by taking the velocity at the edge of the boundary layer. A comparison of these two calculation methods in the Grenoble Workshop by Stanier and Sanchez-Caja [116, 117] showed better agreement near the leading and trailing edge with a significant change in the trailing edge area. The large difference between the two calculation methods is associated with viscous effects near the trailing edge and measurement difficulties at the leading edge [117].

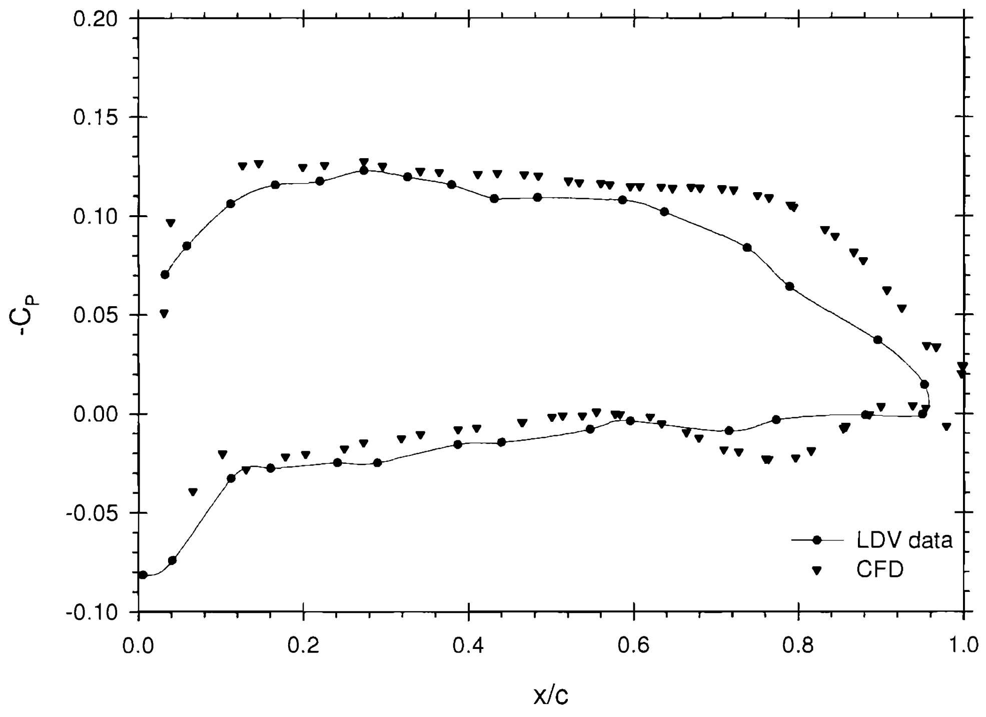


Figure 8.3 – C_P comparison for LDV and CFD at 0.9r/R

8.2.2 Circumferential averaged data

Comparing the average circumferential velocities from the experimental LDV results with the results from the 5d unsteady simulation, we can see that there is very good agreement. The V_x-1 is in particularly good agreement in the vicinity of the blade tip. The point of change in sign of V_x-1 is within 0.01 r/R of the experimental results which is an improvement over all the methods in the Grenoble Workshop [114] which were at 0.025 r/R. This improved agreement is true for V_t and V_r in the vicinity of the blade tip. V_t is over predicted near the hub which was typical of all the methods in the Grenoble Workshop.

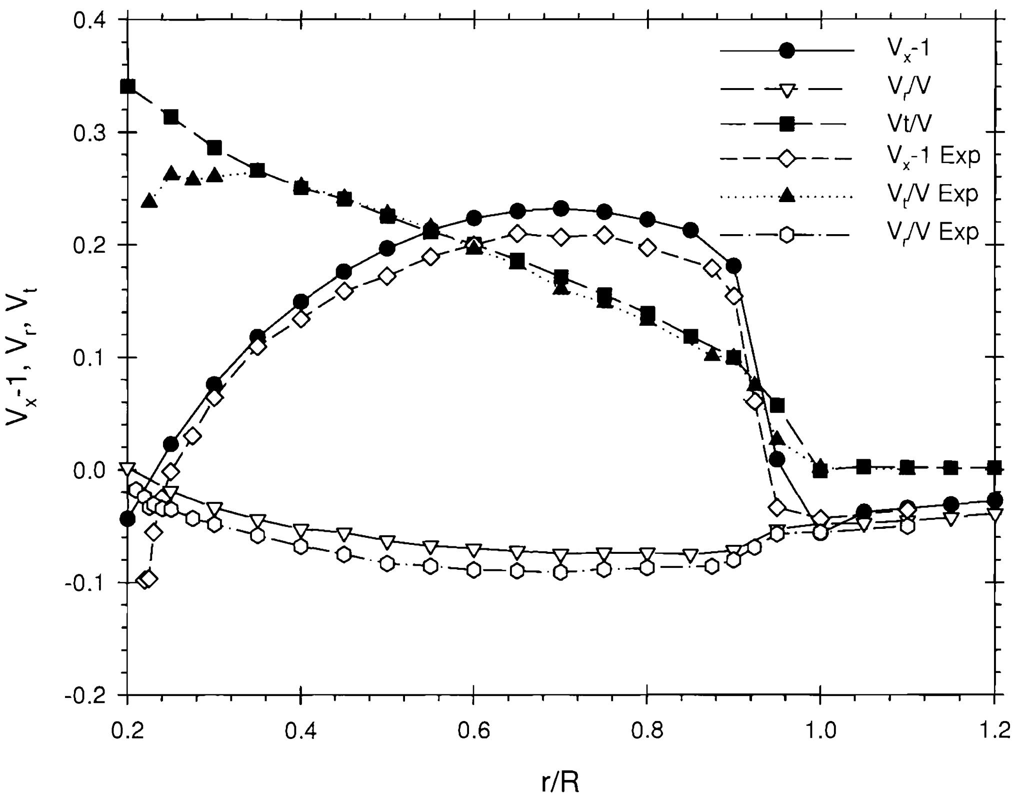


Figure 8.4 – Average circumferential velocity comparison for CFD and LDV

8.2.3 Phase averaged data

The phase averaged data for a plane 0.3281x/R downstream of the propeller is compared with the LDV results at two radii near the tip vortex. The point where V_x-1 and V_r change sign is located at 0.924r/R. Most of the methods at the Grenoble workshop failed to capture the detail in the flow well near the tip vortex. One of the difficulties is that in order to compare the data at a specific radius the tip vortex contraction has to be calculated correctly. If the tip vortex is not at the correct radius then the data will be very different. The Grenoble editors recommended that the analytical data be corrected depending on the tip vortex contraction for better agreement. None of the contributors stated if the results they presented were corrected or not. However the results presented in this work have no correction at all.

From Figure 8.5 & Figure 8.6 we can see an improvement over the results presented in the Grenoble workshop. The tip vortex contraction and position is captured correctly as can be seen from the point of inflection for V_x, V_r and position of the peaks for V_t. For both radii the position of the peaks and inflections are predicted very well. The peak values are over predicted for V_t and V_r and under predicted for V_x. However this is still an improvement over the results presented at the Grenoble workshop, especially for V_r. All the methods substantially under predicted the peak magnitudes for V_r. The best result under predicted the peaks by at least 0.4Vr whereas the results presented here over predict Vr by a similar amount. The best performing method at Grenoble was by Chen and Stern [118]. They used a structured grid with clustering of 20x20 near the estimated tip vortex region. The equations were solved as unsteady with time serving as a convergence parameter. As discussed and shown previously the unsteady solution performs much better with respect to the tip vortex and it is not surprising that this method was the best performing at Grenoble with respect to the tip vortex. The clustering of the grid near the tip vortex was also fundamental to the success. At short distances downstream of the blade it is easy to estimate the tip vortex *a priori*, however further downstream this method would prove impractical and the results would deteriorate if the clustering is not in the region of the tip vortex.

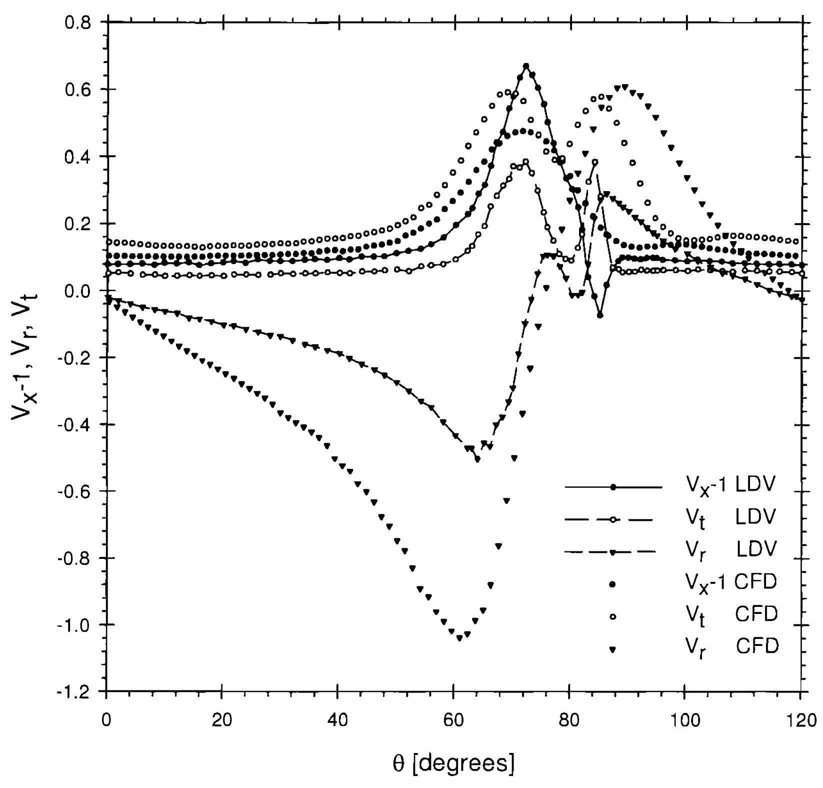


Figure 8.5 – Phase averaged comparison at 0.9r/R and 0.3821x/R

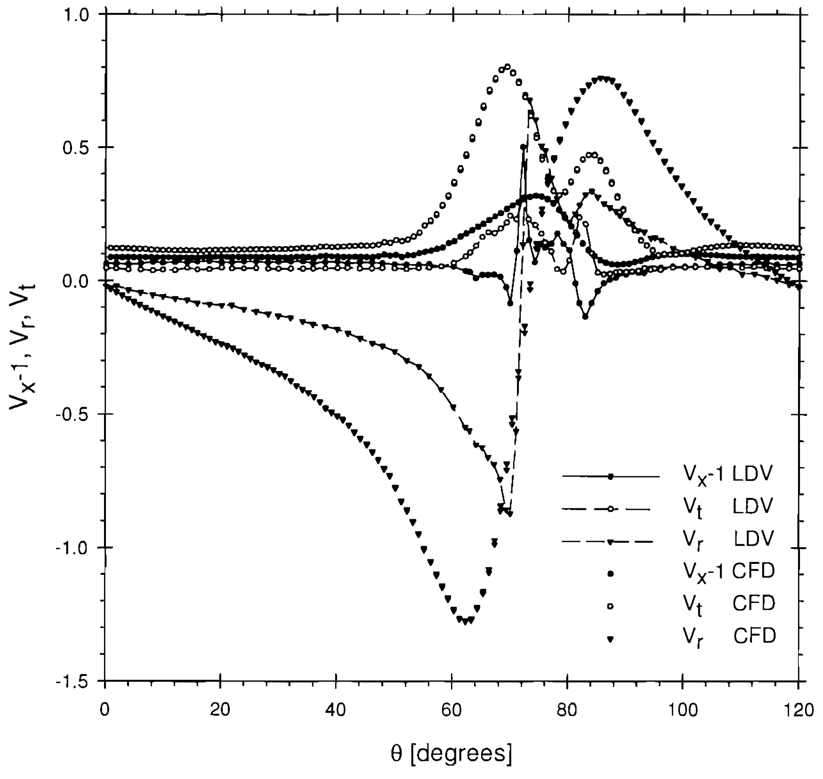


Figure 8.6 - Phase averaged comparison at 0.924r/R and 0.3821x/R

9 INSEAN E779A and numerical model

The INSEAN E799A is a four blade propeller, Wageningen modified type, skewed, with a uniform pitch (P/D 1.1), a forward rake angle of $4^{0}3"$ and a diameter of 227.2 mm. The propeller was designed at the end of the 50's for a twin-screw ferry. In the 60's the model propeller was selected as the reference model for the Italian Navy Cavitation Tunnel (C.E.I.M.M.) where all the measurements were carried out [119].

Extensive tests were performed both in the towing tank and water tunnel and detailed data is available both from LDV [120, 121, 122, 123, 124, 125, 126, 119] and PIV [125, 126, 127, 128] tests.

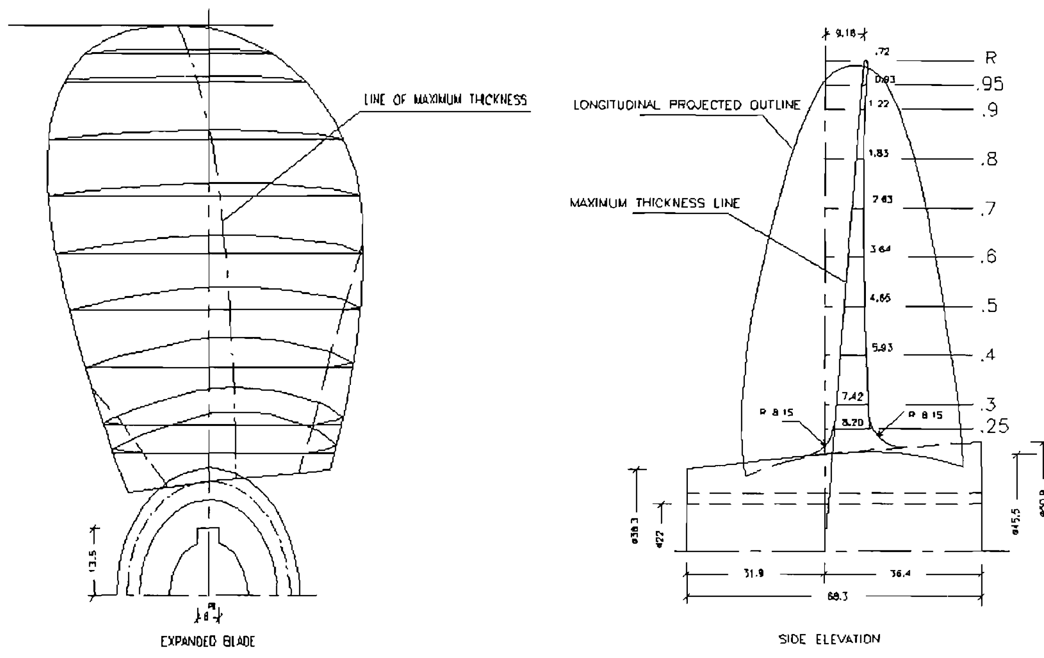


Figure 9.1 – Geometry of the INSEAN E799A four bladed propeller model

The propeller was tested in the C.E.I.M.M. tunnel. The test section is a square, closed jet type with dimensions of 0.6m x 0.6m x 2.6m. The propeller is driven by an upstream shaft. The geometry of the test section was not modelled in the numerical analysis. The test section in the numerical model was circular, like in the DTMB P4119 case. The numerical model was a 90^{0} segment containing one blade of the propeller. Rotational image boundaries were used to model the whole propeller. The outer boundary was a cylinder at a diameter equivalent to 3 times the propeller diameter. The outer boundary was set as a free slip wall. The full geometry of the drive shaft was modelled and extended all the way to the inlet boundary. Supports struts and other protrusions were neglected.

The base mesh for the E799A consisted of 110,013 cells. No clustering of cells was performed apart from near the surfaces of the propeller. In order to simplify the mesh generation due to time restrictions no inflation layer was used near the blade surfaces. Instead the near wall functions were relied upon for near wall modelling with a y^+ value of 100. The mesh was then progressively refined in the region of the tip vortex according to the solution and the VFX method. The maximum cell edge length in the refinement area was progressively decreased for each mesh.

9.1 Results

The refinement procedure was similar to the DTMB P4119. As the mesh is refined the vortex propagates further downstream. The thrust and torque improves with mesh refinement. The thrust is predicted well whereas the torque is under-predicted in the order of 7%. This difference can be attributed to the lack of mesh density in the viscous shear layer since no inflation layer was used as previously explained. The difference for not resolving the boundary and wake adequately is significant as demonstrated for the wing case (5.3).

Three unsteady iterations were carried out each having three timesteps of 0.1s. The previous simulation was used as the starting point for the subsequent run. The results presented are the forces for the final timestep of each simulation. The force prediction showed improvement with increasing number of timesteps. In addition the vortex propagated further downstream.

Mesh	Size	K_T	Δ% K_T exp	K_Q	Δ% K_Q exp
Base	110,013	0.158	-2.7	0.0293	-7.9
Refined1	569,076	0.160	-1.5	0.0298	-6.3
Refined 2	1,067,513	0.159	-1.6	0.0294	-7.4
Refined 3	1,163,111	0.157	-2.9	0.0297	-6.5
Refined 4	1,620,593	0.157	-3.1	0.0293	-8.0
Refined 4 uns		0.161	-0.8	0.0299	-5.9
Refined 5	3,041,667	0.155	-4.2	0.0291	-8.4
Refined 5a uns3		0.156	-3.6	0.0293	-7.9
Refined 5b uns6		0.159	-1.9	0.0295	-7.2
Refined 5c uns9		0.159	-1.8	0.0295	-7.2
Experimental		0.162		0.0318	

Table 9-1 - INSEAN E799A force comparison

9.2 Comparison with LDV data

9.2.1 Circumferential averaged data

Comparing the average circumferential velocities from the experimental LDV results with the results from the 5c unsteady simulation, we can see that there is very good agreement. Compared with the DTMB P4119 the flow acceleration is more uniform from hub to tip. This is also reflected in the LDV data. The V_x-1 shows the correct trend and again is in particularly good agreement in the vicinity of the blade tip with regards to the span position of rapid change in V_x-1. V_t and V_r show agreement over the whole span with the exception being V_r at the blade tip. There is a sudden reduction in LDV data V_r at the blade tip which is not reflected in the CFD results.

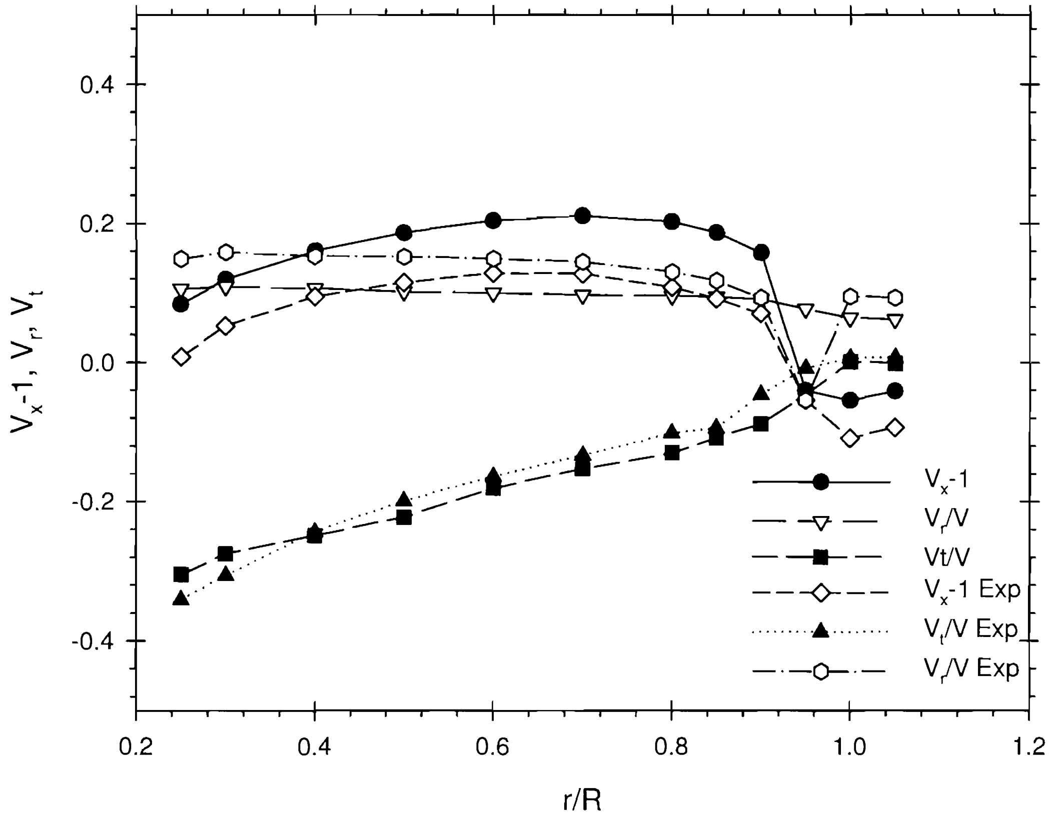


Figure 9.2 – Circumferential averaged data at 0.20 x/R

9.2.2 Contour plot comparison

The contour plots for axial, radial and tangential velocity are compared at 0.20 x/R and 0.65 x/R with the data from the LDV tests. The contour plots are plotted side by side (Figure 9.3 - Figure 9.5). The LDV data is available for all four blades of the propeller. Each blade passage shows slightly different results. Only 180° segments are displayed for the LDV data. The data from the numerical simulation are periodically symmetric; however for viewing purposes an 180° segment is also displayed. The propeller is rotating clockwise in all the contour plots.

From Figure 9.3 it can be seen that there is reasonable agreement with the LDV data, with the position of the tip vortex being predicted very well. The axial velocity deficit for the region of fluid outward of the tip vortex is also captured very well. The accelerated region visible in the LDV data is present in the simulation results; however it is larger than the experimental data. In addition the wake is not very well defined. This is because the mesh resolution in the way of the wake is not adequate. As indicated before (5.3) to capture the wake correctly a small mesh spacing is required, in the order of 20 nodes across the wake. However the position of the wake is predicted well as well as the decelerated flow region near the hub.

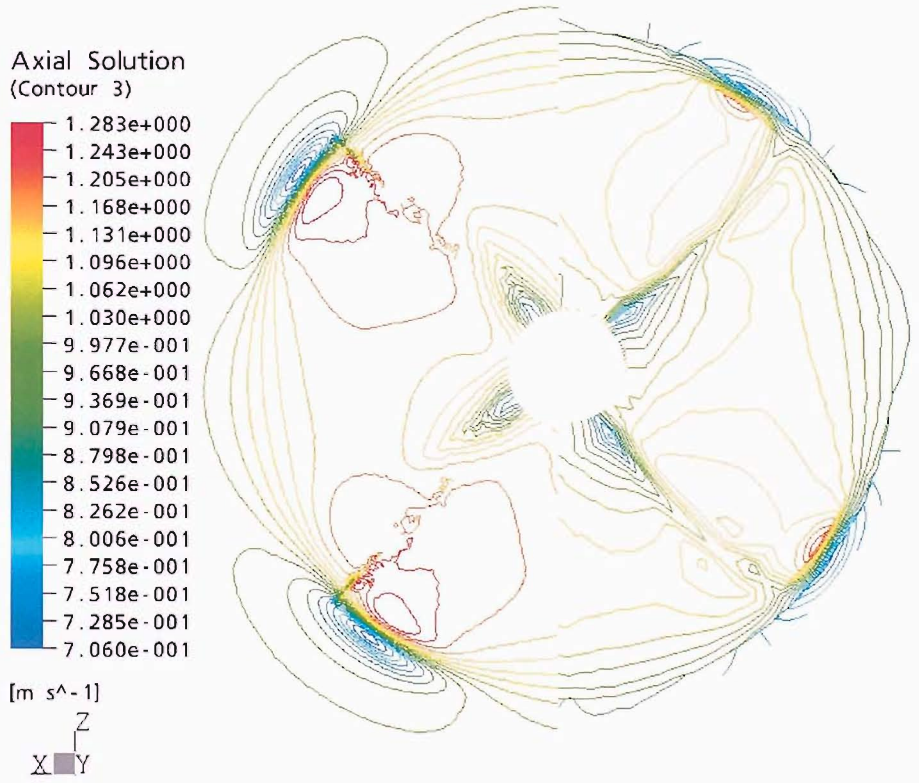


Figure 9.3 – Axial velocity/U contours at 0.20 x/R. CFD left – LDV right

The comparison at 0.65 x/R (Figure 9.4) gives similar observations. There is good agreement in the vortex position and general characteristics. The large deficit in the hub has disappeared as reflected in the LDV data.

The wake roll-up is clearly visible in the LDV data where as it is not apparent in the numerical results. This again is down to the coarse mesh in the vicinity of the wake. As before, the wake velocity deficit is not captured due to this. The wake roll-up and velocity deficit was captured in the wing case where the mesh in the way of the wake was more refined.

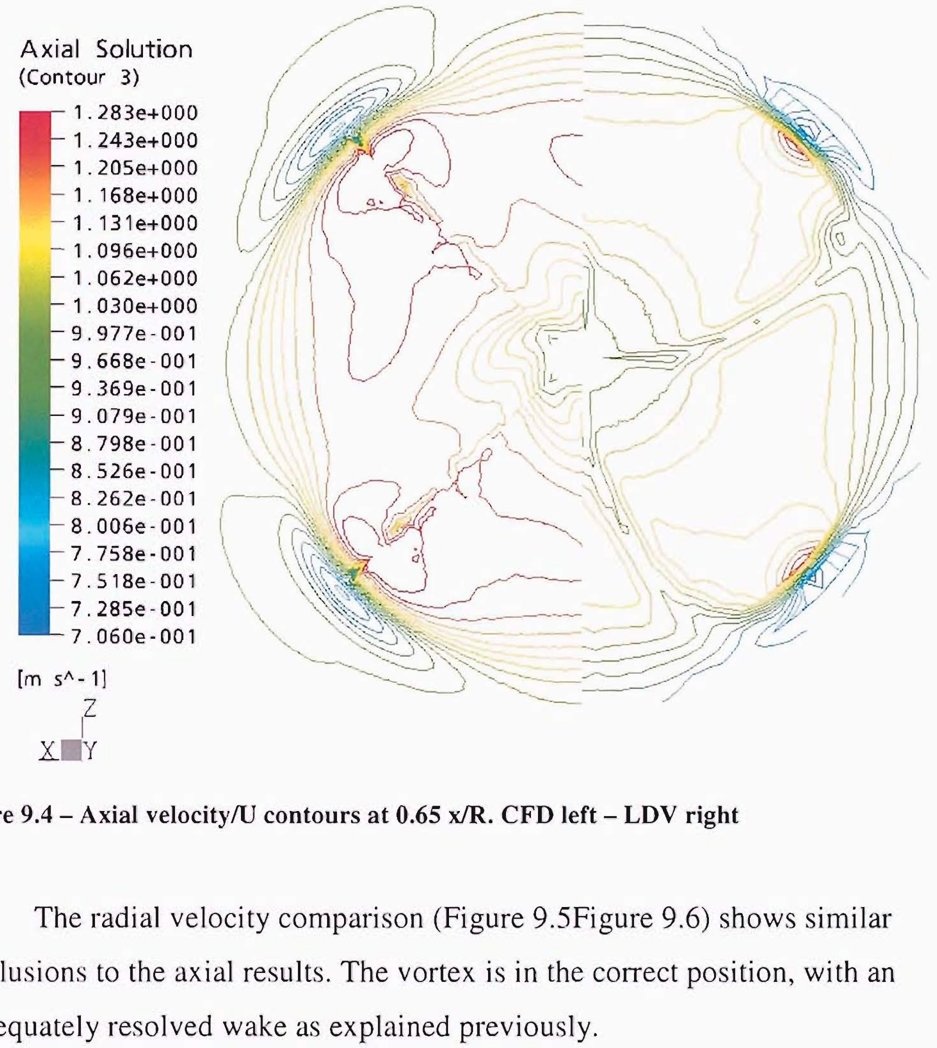


Figure 9.4 – Axial velocity/U contours at 0.65 x/R. CFD left – LDV right

The radial velocity comparison (Figure 9.5Figure 9.6) shows similar conclusions to the axial results. The vortex is in the correct position, with an inadequately resolved wake as explained previously.

The LDV vortex has a smaller diameter than the CFD results. We can see that the vortex cuts across the sampling plane. There is the outward radial velocity region which corresponds to the right hand side of the counter clockwise rotating vortex, followed by the inward radial velocity caused by the opposite side of the vortex (marked by arrows on the CFD results in Figure 9.6).

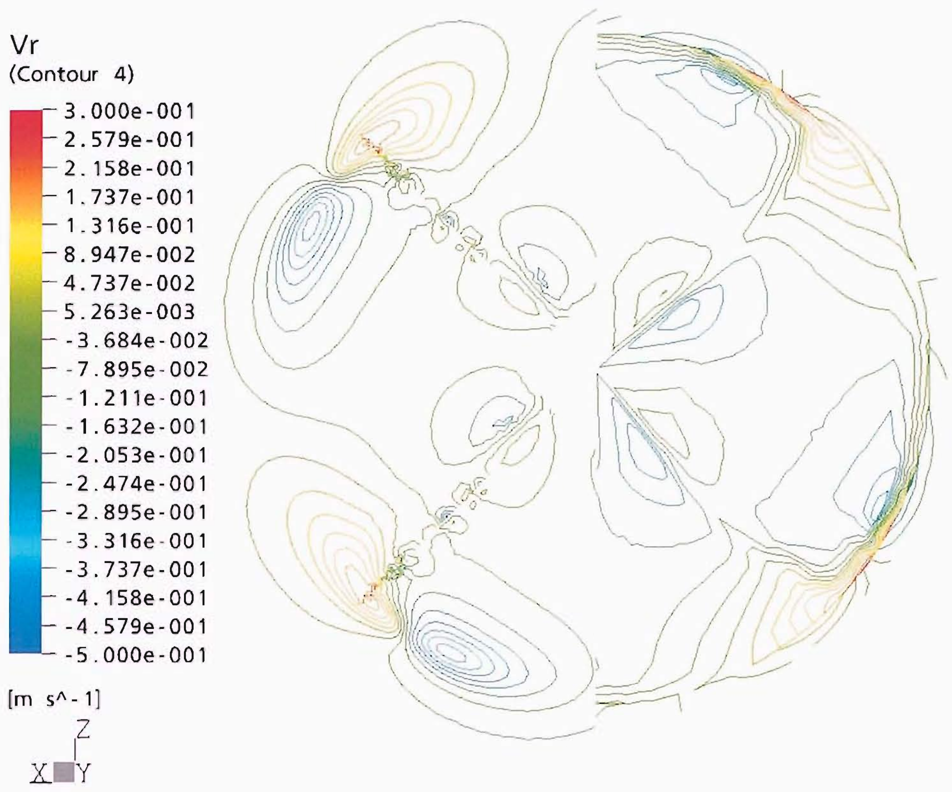


Figure 9.5 – Radial velocity contours at 0.20 x/R. CFD left – LDV right

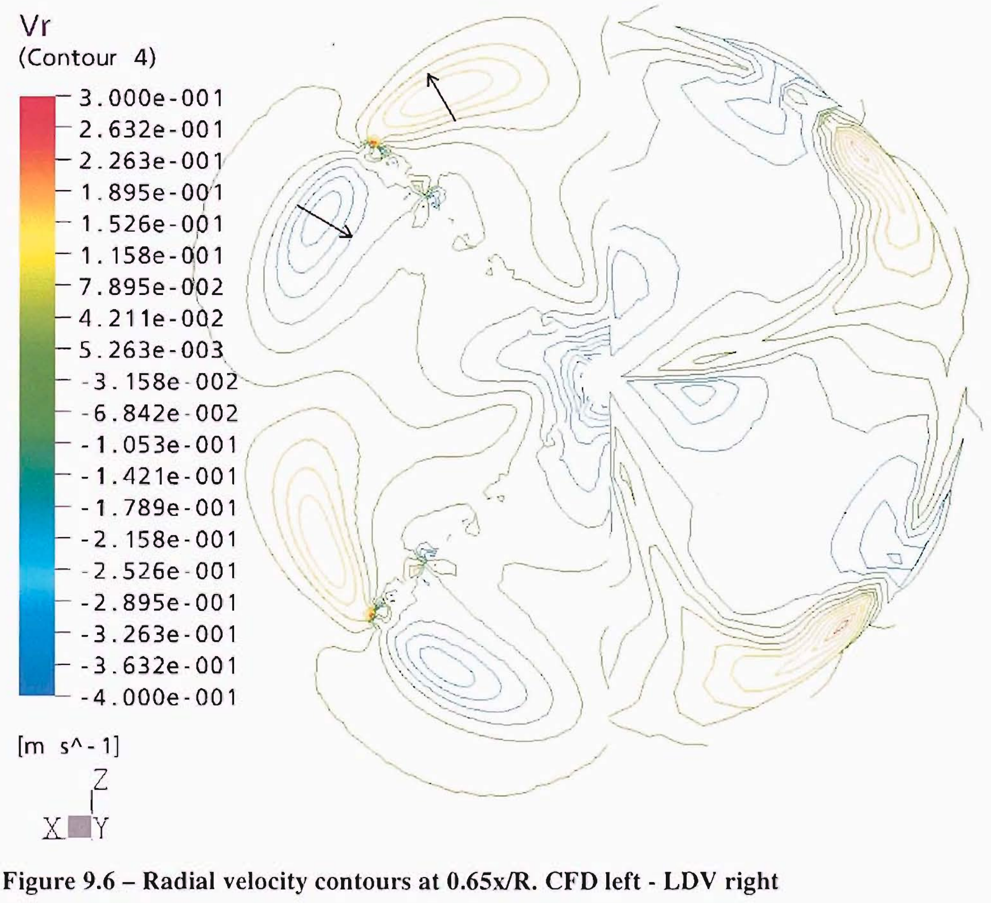


Figure 9.6 – Radial velocity contours at 0.65x/R. CFD left - LDV right

10 Conclusions

The aim of this research was to develop a vortex identification algorithm capable of being used for tip vortex mesh refinement for marine propellers. Such a method has been developed and applied in conjunction with mesh refinement. The scheme successfully identified the tip vortex for a wing and two propeller geometries. The results show an improvement in K_T and K_Q prediction with each mesh refinement. In addition it has been demonstrated that the steady state RANS equations smooth out the tip vortex structure which has difficulty propagating downstream. Using an unsteady formulation of the RANS equations allows the vortex to develop at significantly larger distances downstream of the propeller as observed in experimental tests.

The Vortfind method was modified and refined to make its use with mesh adaptation easier. The method was reformulated such that it can be carried out for an arbitrary number of sectors. In addition the l function was normalised to aid its use with threshold type adaptation. The method was applied to 2D cases to study the effects of the governing parameters, showing good results.

The method was extended for 3-D cases, by the use of planes. The resulting VFX algorithm was initially tested on a wing. The VFX was able to identify the vortex core with great accuracy, within a computational cell of the visual location of the vortex. The VFX method was then used to refine the mesh in the vortex region of the wing. The position of the vortex core was dependant on the mesh density stabilising for the finer grids. The vortex velocity profiles also changed significantly with varying mesh spacing.

The effect of the refinement on the forces and the flow has been clearly demonstrated. In addition simple wake capture algorithm was developed and used to refine the wake using prisms instead of tetrahedral to limit the mesh overhead. Both vortex and wake refinement showed improved agreement with the experimental results.

10.1 Vortex capture

From the results of the mesh study it has been shown that at least15 computational nodes across the vortex core are required to capture the flow of a vortex. This agrees with Dacles and Zilliac [41]. In addition it has been shown that the potential region of the vortex must also be captured correctly in order to get the correct answer. If the mesh resolution in the outer region is not adequate the vortex core will not be correct even if the mesh resolution is adequate.

10.2 Propeller tip vortex

The process was automated and applied to propellers where the method succeeded to track the complex helicoidal blade tip vortex. The starting point specified as the blade tip and the VFX algorithm automatically generated the subsequent planes depending on the vortex core line identified. The propagation of the tip vortex further downstream was found to be very short with each refinement step. To accelerate the propagation the mesh was refined on a predicted vortex core line which had a helicoidal shape with its pitch dependant on the upstream identified tip vortex. This method successfully sped up the vortex propagation downstream typically requiring half the mesh adaptation steps than the normal procedure.

Once more the forces showed improvement over the unrefined mesh when compared to experimental results. Apart from mesh refinement there was a marked improvement in capturing the tip vortex by running a few iterations with an unsteady scheme. Both the vortex dissipation and force discrepancy showed marked reductions.

If the grid is refined enough in the tip vortex region then it is possible to use multiphase models to capture the tip vortex cavitation. If this is modelled successfully it will provide better understanding of the tip vortex physics. Before a solution to this problem can be obtained a suitable mesh must be generated. This work will provide the tools to generate such a refined mesh.

10.3 Structured vs. Unstructured

Structured meshes inherently have the boundary layer region extend downstream in the wake, which give better results. However this does not automatically guarantee better results with structured meshes. For simple cases such as the wings modelled here the vortex position and wake are more or less straight downstream of the wing. In other cases the blocking has to be modified to ensure that the fine mesh lies in the wake and vortex otherwise they will not be captured. This is the case for propellers where the vortex and wake have a helicoidal shape. Stanier reports that wake aligned structured meshes give better results [107]. Even for structured meshes knowing the vortex position can be advantageous in the meshing of future simulations or even with the use of moving meshes.

10.4 Vortex identification methods on planes

Sujudi and Haimes [63] stated that plane methods were laborious. This is not entirely true. It has been shown here that plane methods can easily be implemented even for a vortex which has a helicoidal path. In addition, the computational effort is substantially reduced due to the limited area of over which the method works. However as the number of vortices increases plane methods do become more difficult to implement. The method developed here can handle multiple vortices as long as their axes are within 30-40° of each other since the Vortfind method is relatively insensitive to plane alignment with the vortex axis [1]. For multiple vortices that have their axis at larger angles the method will have to be extended such that there is starting plane for each vortex. Then using the tracking procedure described for the propeller on each individual vortex they can be identified without any restrictions on the vortex paths.

10.5 Final Remark

The vortex identification scheme developed here is not limited to tip vortex flows. It is a potentially powerful tool applicable to all swirling flows and the author hopes that future researchers will extend, refine and use this tool for future research.

References

1 PEMBERTON RJ, "*A vortex identification technique for grid adaptation*", PhD thesis, School of Engineering Sciences, Ship Science, 2003

2 PEMBERTON RJ, TURNOCK SR, DODD TJ, ROGERS E, "*A novel method for identifying vortical structures*", Journal of Fluids and Structures, 16(8), pp. 1051-1057, 2002

3 RINA , "*World's largest composite propeller successfully completes ship trials*", The Naval Architect July/August, 2003

4 ARNDT REA, "*Cavitation in vortical flows*", Annu. Rev. Fluid Mech, 34:pp143-175, 2002

5 ITTC, "*The specialist committee on cavitation induced pressures: Final report and recommendations to the 23rd ITTC*", Proceedings 23rd ITTC, Vol.II, 2002

6 KUIPER G, "New developments around sheet and tip vortex cavitation on ships' propellers", CAV2001: 4th International Symp. On Cavitation, Pasadena, 2001

7 ANDERSEN P, BRESLIN JP, "*Hydrodynamics of ship propellers*", Cambridge Univ. Press., 1995

8 LEWIS EV, "*Resistance, Propulsion and Vibration*", Principles of Naval Architecture, Volume II, SNAME, pp.165, 1988

9 RANKINE WJM, "*A manual of civil engineering*" Charles Grifin and Co, 4th edition, 1865

10 TAYLOR DW, "S*ome aspects of the comparison of model and full-scale tests*", NACA, Report 219

11 BETZ A, "*Schraubenpropeller mit geringstem Energieverlust*", K. Ges. Wiss, Gottingen Nachr. Math.-Phys. pp193-217, 1919

12 GOLDSTEIN S, "*On the vortex theory of screw propellers*", Proc. R. Soc. London Ser. A 123:440-465, 1929

13 PRANDTL L, "*Application of modern hydrodynamics to aeronautics*", NACA Annual Report, 7th, pp 157-215, 1921

14 LERBS HW, "*Moderately loaded propellers with a finite number of blades and an arbitrary distribution of circulation*", SNAME Trans. 60:73-117, 1952

15 SPARENBERG JA, "*Application of lifting surface theory to ship screws*", Proc. K. Ned. Akad. Wet.-Asterdam, Ser. B62(5), pp286-298, 1959

16 PIEN PC, "*The calculation of marine propellers based on lifting surface theory*", J. Ship Research, 5(2), pp1-14, 1961

17 KERWIN JE, "*The solution of propeller lifting surface problems by vortex lattice methods*", Report Department Ocean Engineering MIT, 1979

18 VAN MANEN JD, BAKKER AR, "*Numerical results of Sparenberg's lifting surface theory of ship screws*", Proc. 4th Symposium on Naval Hydrodynamics, pp63-77 Washington, 1962

19 ENGLISH JW, "*The application of a simplified lifting surface technique to the design of marine propellers*", Ship Div. Natl. Phys. Lab., Fetlham, England, 1962

20 BROCKETT TE, "*Lifting surface hydrodynamics for design of rotating blades*", Proc. SNAME Propellers '81 Symp., Virginia, 1981

21 GREELEY DS, KERWIN JE, "*Numerical methods for propeller design and analysis in steady flow*", SNAME Trans. 90:415-453, 1982

22 HESS JL, VALAREZO WO, "*Calculation of steady flow about propellers by means of a surface panel method*", AIAA Paper No. 85-0283, 1985

23 HESS JL, "*Review of integral-equation techniques for solving potential-flow problems with emphasis on the surface-source method*", Comput. Methods Appl. Mech. Eng., 5, pp.145-196, 1975

24 PASHIAS C, TURNOCK SR, ABU-SHARKH S, "*Design optimisation of a bi-directional integrated thruster*", SNAME Propeller/Shafting Symposium 03, Virginia Beach USA, Sept 2003

25 KIM HT, STERN F, "*Viscous flow around a propeller-shaft configuration with infinite pitch rectangular blades*", Journal of Propulsion and Power, Vol.6, No.4, pp.434-444, 1990

26 UTO S, "*Computation of incompressible viscous flow around a marine propeller*", Journal of SNAJ, Vol 172, pp.213-224, 1992

27 STANIER MJ, "*Design and evaluation of new propeller blade sections*", 2nd International STG Symp. On Propulsors and Cavitation, Hamburg, Germany, 1992

28 CHEN B, STERN F, KIM WJ, "*Computation of unsteady marine propulsor blade and wake flow*", Proc 20th ONR Symposium on Naval Hydrodynamics, Santa Barbara, 1994

29 ABDEL-MAKSOUD M, MENTER FR, WUTTKE H, "*Numerical computation of the viscous flow around series 60 C_B=0.6 ship with rotating propeller*", Proc. 3rd Osaka Colloquium Advanced CFD Applications to Ship Flow and Hull Form Design, pp.25-50, Osaka, 1998

30 STANIER MJ, "*Numerical prediction of propeller scale effect*", PhD. Thesis, University of Southampton, 2001

31 BULL P, "*The validation of CFD predictions of nominal wake for the SUBOFF fully appended geometry*", 21st ONR Symposium on Naval Hydrodynamics, pp1061-1076, 1996

32 ITTC, "*The propulsion committee: Final report and recommendations to the 22nd ITTC*", Proceedings 22nd ITTC, Seoul and Shanghai, 1999

33 MARCHAJ CA, "*Aero-hydrodynamics of sailing*", 2nd Edition, Adlard Cole Nautical, 1979

34 THEODORE TA, "*Introduction to the aerodynamics of flight",* SP-367, Scientific and Technical Information Office, National Aeronautics and Space Administration, Washington, D.C. 1975

35 KRAMER M, "*Boundary layer stabilization by distributed damping*", Journal of Aerospace Science 24, 1957

36 LUGT HJ, "*Vortex flow in nature and technology*", New York, John Wiley & Sons, 1983

37 LANCHESTER F, "*Aerial flight*, Vol. I: Aerodynamics", London: Constable, 1907.

38 PRANDTL L., "*Essentials of fluid dynamics*", Blackie & Son, 1952

39 PRANDTL L, TIETJENS OG, HARTJOG J, "*Applied hydro and aeromechanics.*" London, England: McGraw-Hill Book Company, Inc., 1934.

40 SPALART PR, "*Airplaine trailing vortices*", Annu. Rev. Fluid Mech. 30:107-138, 1998

41 DACLES-MARIANI J, ZILLIAC GG, "*Numerical/Experimental study of a wingtip vortex in the near field*", AIAA Journal, Vol.33, No.9 pp.1561-1568, 1995

42 VIOT X, FRUMAN D, DENISET F, BILLARD J, "*Numerical simulation of tip vortices roll-up*", 22nd Symposium on Naval Hydrodynamics, 2000

43 SPALL RE, "*Numerical study of a wing-tip vortex using the Euler equations*", Journal of Aircraft, Vol.38, No.1, 2001

44 BERNTSEN GS, KJELDSEN M, ARNDT REA, "*Numerical modelling of sheet and tip vortex cavitation with Fluent 5*", CAV2001: session B5.006, 2001

45 HSIAO CT, CHANINE GL, "*Scaling of tip vortex cavitation inception noise with a bubble dynamics model accounting for nucleus size distribution*", International Symp. On Cavitation Inception, Honolulu, 2003

46 MAVRIPLIS DJ, "*Unstructured grid techniques*", Annu. Rev. Fluid. Mech., 29:473–514, 1997

47 MULLER JD, GILES MB, "*Solution adaptive mesh refinement using adjoint error analysis*", AIAA Paper 2001-2550, 2001

48 NITHIARASU P, ZIENKIEWICZ OC, "*Adaptive mesh generation for fluid mechanics problems*", Int. J. Numer. Meth. Engng. 47:629-662, 2000

49 MAVRIPLIS DJ, "*Adaptive meshing techniques for viscous flow calculations on mixed element unstructured meshes*", Int. J. Numer. Meth. Fluids, 34: 93–111, 2000

50 BOTTASSO CL, SHEPHARD MS, "*Finite element adaptive multigrid euler solver for rotary wing aerodynamics*", AIAA Journal, Vol.38, No.1, pp 50-56, 2000

51 LÖHNER R, "*Mesh adaptation in fluid mechanics*", Engineering Fracture Mechanics, Vol.50, No.5/6, pp.819-847, 1995

52 PELLETIER D, "*Adaptive finite element computations of complex flows*", Int. J. Numer. Meth. Fluids 31: 189–202, 1999

53 BARTH TJ, "*Aspects of unstructured grids*", VKI Lecture Series, 1994-05, revised February 1995

54 ROGERS SE, ROTH K, NASH SM, BAKER MD, SLOTNICK JP, WHITLOCK M, CAO HV, "*Advances in overset CFD processes applied to subsonic high-lift aircraft*", 18th AIAA Applied Aerodynamics Conference, AIAA 2000-4216, Colorado, August 2000

55 CHAN WM, GOMEZ III RJ, ROGERS SE, BUNING PG, "*Best practises in overset grid generation*", 32nd AIAA Fluid Dynamics Conference, AIAA 2002-3191, Missouri, June 2002

56 DINDAR M, SHEPHARD MS, FLAHERTY JE, JANSEN K, "*Adaptive CFD analysis for rotorcraft aerodynamics*", Comput. Methods Appl. Mech. Engrg. 189: 1055-1076, 2000

57 LEVY Y, DEGANI D, SEGINER A, "*Graphical visualization of vortical flows by means of helicity*", AIAA Journal, Vol.28, No.8, pp. 1347-1352 , August 1990

58 STRAWN RC, KENWRIGHT DN, AHMAD J, "*Computer visualisation of vortex wake systems*", AIAA Journal, Vol.37, No.4, pp. 511-512, April 1999

59 SADARJOEN IA, POST FH, MA B, BANKS DC, PAGENDARM H-G, "*Selective visualization of vortices in hydrodynamic flows*", IEEE Visualization '98, pp.419-422, October 1998

60 BERDAHL CH, THOMPSON DS, "*Eduction of swirling structure using the velocity gradient tensor*", AIAA Journal, Vol.31, No.1, pp. 97-103, August 1990

61 JEONG J, HUSSAIN F, "*On the identification of a vortex*", Journal of Fluid Mechanics, 285:69-94, 1995

62 BANKS D, SINGER B, "*A predictor-corrector technique for visualizing unsteady flow*", IEEE Transactions Visualization and Computer Graphics 1:151-163,1995

63 SUJUDI D, HAIMES R, "*Identification of swirling flow in 3-D vetor fields*", AIAA 12th Computational Fluid Dynamics Conference, Paper 95-1715, June 1995

64 ROTH M, PEIKERT R, "*A higher-order method for finding vortex core lines*", Proceedings of the conference on Visualization '98, North Carolina, 1998

65 JIANG M, MACHIRAJU R, THOMPSON D, "*A novel approach to vortex core region detection*", Joint Eurographics – IEEE TCVG Symposium on Visualization, pp.217-225, May 2002

66 JIANG M, MACHIRAJU R, THOMPSON D, "*Detection and visualization of vortices*", Department of Computer and Information Science, Ohio State University, 2002

67 GODO M, SCHECTERMAN M, LEGENSKY S, HAIMES R, "*Applicability of vortex cores to CFD simulations with realistic geometry models*", 39th AIAA Aerospace Sciences Meeting and Exhibit, AIAA-2001-0914, Reno, January 2001

68 JIANG M, MACHIRAJU R, THOMPSON D, "*Geometric verification of swirling features in flow fields*", IEEE Visualization '02, pp.307-314, October 2002

69 MURAYAMA M, NKAHASHI K, SAWADA K, "*Simulation of vortex breakdown using adaptive grid refinement with vortex-centre identification*", AIAA Journal, Vol.39, No.7, 2001

70 KENWRIGHT D, HAIMES R, "*Vortex identification - applications in aerodynamics: A case study*", Proc. of the 8th conference on Visualization '97, Phoenix, Arizona, United States, 1997

71 CEBRAL JR, LÖHNER R, "*Flow visualization on unstructured grids using geometrical cuts, vortex detection and shock surfaces*", 39th AIAA Aerospace Sciences Meeting and Exhibit, AIAA-2001-0915, Reno, January 2001

72 HENTSCHEL R, "*The creation of lift by sharp-edged delta wings. An analysis of a self-adaptive numerical simulation using the concept of vorticity content*", Aerospace Science and Technology, No.2:79-90, 1998

73 MURAYAMA M, NAKAHASHI K, SAWADA K, "*Adaptive grid refinement coupled with vortex core identification for vortex flow simulation about a delta wing*", Proceedings of International Symposium on Computational Fluid Dynamics, Bremen, Germany, September 1999

74 MURAYAMA M, NAKAHASHI K, OBAYASHI S, "*Numerical simulation of vortical flows using vorticity confinement coupled with unstructured grid*" AIAA-2001-0606, 39th Aerospace Sciences Meeting and Exhibit, Reno, U.S.A., January 2001

75 MURAYAMA M, NAKAHASHI K, OBAYASHI S, "*Simulation of wing tip vortices using vorticity confinement on unstructured grid*" Proc. JSASS 14th International Sessions in 38th Aircraft Symposium, Sendai, Japan, October 2000.

76 SHEN YT, "*USS Bulkeley (DDG 84) Twisted rudder coordinated trial results*", Report No. NSWCCD-50-TR-2000/056

77 JESSUP S, "*Experimental data for RANS calculations and comparisons (DTMB4119)*" 22nd ITTC Propulsion Committee, Propeller RANS/Panel Method Workshop, Grenoble, France, April 1998

78 HSIAO C-T, PAULEY LL, "*Numerical computation of tip vortex flow generated by a marine propeller*", Transactions of the ASME, Vol.121: 638-645, 1999

79 LEE Y-T, HAH C, LOELLBACH J, "*Investigation of tip clearance vortex structures through numerical flow visualization*", ASME Fluids Engineering Division Conference, FED Vol.229: 157-165, 1996

80 OH K-J, KANG S-H, "*Numerical calculation of the viscous flow around a propeller shaft configuration*", International Journal of Numerical Methods in Fluids, Vol 21:1-13, 1995

81 ABDEL-MAKSOUD M, HELLWIG KATRIN, BLAUROCK J, "*Numerical and experimental investigation of the hub vortex flow of a marine propeller*", 25th Symposium on Naval Hydrodynamics, St. Johns, 2004

82 FERZIGER JH, PERIC MP, "*Computational methods for fluid dynamics*", 2nd edition, Springer, 1999

83 ANDERSON, "*Computational methods for fluid dynamics: Basics with applications*", McGraw-Hill Inc., 1995

84 ***Fluent v6.1***, FLUENT INC, 10 Cavendish Court, NH 03766, USA

85 ***CFX-5.1***, ANSYS INC., 275 Technology Drive, PA15317, USA

86 RHIE CM, CHOW WL, "*A numerical study of the turbulent flow past an isolated aerofoil with trailing edge separation*", AIAA Paper 82-0998, 1982

87 BARTH TJ, JESPERSON DC, "*The design and application of upwind schemes on unstructured meshes*", AIAA Paper 89-0366, 1989.

88 ANSYS INC., "*Solver theory*", CFX-5 Users Manual, 2000

89 RAW M, "*Robustness of coupled algebraic multigrid for the Navier-Stokes equations*", AIAA 96-0297, 34th Aerospace and Sciences Meeting & Exhibit, January 15-18, Reno, NV, 1996

90 HUTCHINSON BR, RAITHBY GD, "*A multigrid method based on the additive correction strategy*", Numerical Heat Transfer, Vol. 9, pp. 511-537, 1986.

91 OSAMA AK, TIN-CHEE W, IHAB A, LIU CH, *"Prediction of near- and far-field vortex-wake turbulent flows"*, AIAA Atmosperic Flight Mechanics Conference, Balitimore August, AIAA 95-3470-CP, 1995

92 WALLIN S, GIRIMAJI SS, *"Evolution of an isolated turbulent trailing vortex"*, AIAA Journal, Vol.38, No.4, April, 2000

93 DIETZ W, FAN M, STEINHOFF J, WENREN Y, *"Application of vorticity confinement to the prediction of the flow over complex bodies"*, AIAA CFD conference, Anaheim, CA, AIAA-2001-2642, June, 2001

94 LÖHNER R, YANG C, ROGER R, *"Tracking vortices over large distances"*, 24th Symposium on Naval Hydrodynamics, Fukuoka, Japan, July, 2002

95 ARAKAWA C, FLEIG O, IIDA M, SHIMOOKA M, *"Numerical approach for noise reduction of wind turbine blade tip with Earth Simulator"*, Journal of the Earth Simulator, Vol.2, 11-33, March 2005

96 MENTER FR, *"Two-equation eddy-viscosity turbulence models for engineering applications"*, AIAA Journal, 32(8), 1994

97 PATTENDEN RJ, TURNOCK SR, PASHIAS C, "*Oblique ship flow predictions using identification of vortex centres to control mesh adaptation*", Proc. of CFD Workshop, pp 454-459 , Tokyo, 2005

98 PATTENDEN RJ, TURNOCK SR, BISSUEL M, PASHIAS C, "*Experiments and numerical modelling of the flow around the kvlcc2 hullform at an angle of yaw*", Proc of 5th Osaka Colloquium on advanced research on ship viscous flow and hull form design, pp163-170, 2005

99 SPATH, "*Cluster algorithm analysis for data reduction and classification of objects*", Ellis Horwood Ltd, Chichester, 1980

100 YIN X, GERMAY N, "*A fast genetic algorithm with sharing scheme using cluster methods in multimodal function optimisation*", Proc. of the Int. Conf. on Artificial Neural Nets and Genetic Algorithms, pp450-457, 1993

101 TURNOCK SR, "*Prediction of ship-rudder interaction using parallel computations and wind tunnel measurements*", University of Southampton, Ph.D Thesis, 1993

102 ***ICEM 4 CFD v5.0***, ICEM CFD ENGINEERING, 2855 Telegraph Avenue, CA94705, USA

103 TURNOCK SR, QUERARD ABG, "*Measurements of the evolution of the wake and tip vortex of a control surface undergoing periodic motion*", 27th American Towing Tank Conference, St. John's, Canada, August 2004

104 SANCHEZ-CAJA A, RAUTAHEIMO P, SIIKONEN T, "*Simulation of incompressible viscous flow around a ducted propeller using a RANS equation solver*", 23rd Symposium on Naval Hydrodynamics, 2001

105 WS ATKINS CONSULTANTS AND MEMBERS OF THE NSC, "*Best practice guidelines for marine applications of computational fluid dynamics*," URL: http://pronet.wsatkins.co.uk/marnet/guidelines/guide.html/ [15 Dec 2002].

106 ITTC, "R*eport of resistance and flow committee*", 20th ITTC San Francisco, 1993

107 STANIER M, "*Numerical prediction of propeller scale effect*", University of Southampton, Ph.D Thesis, 2001

108 KERWIN JE, "*Marine propellers*", Ann. Rev. Fluid Mech., 18:367-403, 1986

109 PASHIAS C, TURNOCK SR, "*Hydrodynamic design of a bi-directional rim-driven thruster*", Ship Science report 128, University of Southampton, 2003

110 KOUH JS, LIANG WY, "*Development of a computer aided propeller design system*", Journal SNAME, Vol.14(2), pp.1-17, 1995

111 KOUH JS, HAN CH, CHEN YJ, "*A new geometric modelling method for marine propellers*", 9th Symp. on Practical Design of Ships and other Floating Structures, Luebeck, 2004

112 RYCROFT NC, TURNOCK SR,"*Three dimensional multiblock grid generation: Fleximesh*", Ship Science Report 101, University of Southampton, November 1997

113 HALL CA, "*Construction of curvilinear co-ordinate systems and applications to mesh generation*" International Journal for Numerical Methods in Engineering, 7:461-477, 1973.

114 GINDROZ B, HOSHINO T, PYLKKANEN JV, editors, "*Propeller RANS/Panel method workshop proceedings*" 22nd ITTC Propulsion Committee, Grenoble, 1998

115 BULTEN NWH, OPREA IA, "*Consideration on deviations in torque prediction for propellers and waterjets with RANS codes*", Marine CFD 2005, pp.79-87, Southampton, 2005

116 STANIER MJ, "*The application of a RANS code to model propeller DTRC4119*", 22nd ITTC Propulsion Committee Propeller RANS/PANEL Method Workshop, Grenoble, April, 1998

117 SANCHEZ-CAJA A, "*P4119 RANS calculations at VTT*", 22nd ITTC Propulsion Committee Propeller RANS/PANEL Method Workshop, Grenoble, April, 1998

118 CHEN B, STERN F, "*RANS simulation of marine-propulsor P4119 at design condition*", 22nd ITTC Propulsion Committee Propeller RANS/PANEL Method Workshop, Grenoble, April, 1998

119 CALCAGNO G, DI FELICE F, FELLI M, FRANCHI S, PEREIRA F, SALVATORE F, "*The INSEAN E779a propeller test case: a database for CFD validation*", INSEAN (Italian Ship Model Basin), Rome, Italy

120 CENEDESSE A, ACCARDO L, MILONE R, "*Phase sampling technique in the analysis of a propeller wake*", First Intern. Conf. on Laser Anemometry: Advances and applications, BHRA Fluid Engineering, Manchester, 1985

121 CENEDESSE A, ACCARDO L, MILONE R, "*Phase sampling in the analysis of a propeller wake*", Experiments in Fluids 6, pp.55-60, 1988

122 STELLA A, GUJ G, DI FELICE F, ELEFANTE M, "*Propeller wake evolution analysis by LDV*", 22nd Symp. On Naval Hydrodynamics, 2000

123 STELLA A, GUJ G, DI FELICE F, ELEFANTE M, "*Propeller wake flow field analysis by means of LDV phase sampling techniques*", Experiments in Fluids

124 STELLA A, GUJ G, DI FELICE F, ELEFANTE M, "*Experimental investigation of propeller wake evolution by means of LDV and flow visualization*", Journal of Ship Research, 2000

125 FELLI M, DI FLORIO D, FELICE F, CALACGNO G, "*Propeller wake visualization by laser anemometry*", 6th Asian Symposium on Visualization, Pusan, South Korea, 2001

126 FELLI M, DI FLORIO D, FELICE F, "*Comparison between PIV and LDV techniquesin the analysis of a propeller wake*", Journal of Visualization, Vol.5, n.3, 2002

127 DI FELICE F, ROMANO G, ELEFANTE M, "*Propeller wake analysis by means of PIV*", 23rd Symposium on Naval Hydrodynamics, 2000

128 GIORDANO G, ROMANO GP, COSTANZO M, DI FELICE F, SOAVE M, "*Propeller wake velocity and pressure fields*", HSMV, Napoli, 2002

11 Appendix A

Propgen Source code

```
Option Explicit

Type section
x As Single
y As Single
z As Single
End Type

Type prop
radius As Single
chord As Single
rake As Single
skew As Single
pitch As Single
thickness As Single
End Type

Type section_info
thickness As Single
position As Single
le_id As Integer
te_id As Integer
num_points As Integer
End Type

Public working_path As String

'flag for volume mesh
Public volume_mesh As Integer

Public va As Single 'advance speed
Public rps As Single 'rev per second
Public no_of_wake_points As Integer
Public final_section() As section
Public section() As section
Public section_data() As section_info
Public propdata() As prop
Public num_sections As Integer
Public num_duct_arc_points As Integer
Public blade_true, ring_true As Integer
Public pi As Double
Public num_leading_lower_points, num_trailing_lower_points, num_upper_points As
Integer
Public num_edge_section_points As Integer
Public wake() As section
Public D As Single

'cap variables
Public cap_section_points As Integer
Public cap_auto As Integer
Public cap_sidenode_id As Integer
Public cap_left_temp() As section
Public cap_left() As section
Public cap_right() As section
Public cap_back() As section
Public cap_internal_y() As section
Public cap_internal_x() As section
Public cap_internal_z() As section
Public cap_draw As Integer
Public num_internal_x_points As Integer
Public num_internal_points As Integer
Public cap_panels_s As Integer
Public cap_panels_t As Integer
Public internal_fraction As Single
Public side_fraction As Single


Public wake_draw As Integer

'Hub variables
Public hub_offset_le As Integer
Public hub_v_leading_factor As Single
Public hub_v_trailing_factor As Single
Public hub_length As Single
Public hub_draw As Integer
Public hub_trailing_arc_connect() As section
Public hub_leading_arc_connect() As section
Public hub_trailing_edge() As section
Public hub_leading_edge() As section
Public hub_leading_end() As section
Public hub_trailing_end() As section
Public num_hub_section_points As Integer
Public hub_section_edge() As section
Public no_of_hub_strips As Integer
Public hub_helix_le() As section
Public hub_helix_te() As section
Public hub_helix_blades() As section
Public hub_panels_s, hub_panels_t As Integer
Public test() As section

'duct variables
Public duct_Q As Single
Public duct_P As Single
Public duct_thickness As Single
Public duct_images As Integer
Public duct_lower() As section
Public duct_trailing_lower() As section
Public duct_leading_lower() As section
Public duct_upper() As section
Public duct_leading_arc() As section
Public duct_trailing_arc() As section
Public duct_leading_lower_arc() As section
Public duct_trailing_lower_arc() As section
Public duct_draw As Integer
Public duct_wake() As section
Public duct_wake_arcs() As section
Public duct_length As Single
Public duct_panels_t, duct_upper_panels_s As Integer
Public duct_freewake_length As Single
Public duct_freewake_panels_s, duct_fixedwake_panels_s As Integer
Public duct_wake_length As Single
Public duct_leading_lower_panels_s As Single
Public duct_trailing_lower_panels_s As Single

'blade variables
Public transition_length As Single
Public wake_contraction_value As Single
Public blade_tip_cluster As Single
Public blade_P As Single
Public blade_Q As Single
Public blade_panels_s, blade_panels_t, blade_fixedwake_panels_s,
blade_freewake_panels_s As Integer
Public blade_wake_length, blade_freewake_length As Single
Public no_of_blades As Integer
Public leading_spline() As section
Public trailing_spline() As section
Public wake_pitch_set As Single


'ring variables
'Public second_ring_leading_end() As section
'Public second_ring_trailing_end() As section
'Public second_ring_section_edge() As section
Public ring_panels_t As Integer
Public ring_width As Single
Public ring_split As Integer
Public ring_trailing_edge() As section
Public ring_leading_edge() As section
Public ring_leading_end() As section
Public ring_trailing_end() As section
Public num_ring_section_points As Integer
Public ring_section_edge() As section
Public no_of_ring_strips As Integer
Public ring_helix_le() As section
Public ring_helix_te() As section
Public ring_helix_blades() As section

Public Sub read_working_path()
Open "path.txt" For Input As #1
Input #1, working_path$
```

```
Close #1
End Sub
Public Sub hub_leading_section(hub_length)
Dim num_edge_points, i, j As Integer
Dim psi, dpsi, r, P As Single
Dim angle, angle_step As Single
Dim temp() As section

num_edge_points = 100
ReDim hub_leading_end(10, num_edge_points)
ReDim temp(num_edge_points)

'set point to trailing edge of first section
hub_leading_end(1, 1).x = final_section(1, section_data(1).le_id).x
hub_leading_end(1, 1).y = final_section(1, section_data(1).le_id).y
hub_leading_end(1, 1).z = final_section(1, section_data(1).le_id).z

r = Sqr(hub_leading_end(1, 1).z ^ 2 + hub_leading_end(1, 1).x ^ 2)


P = propdata(1).pitch * D
If P = 0 Then P = 0.1
'calculate the start angle and the step angle

'calculate start psi so it matches with leading edge of section
psi = (final_section(1, section_data(1).le_id).y / P) * 2 * pi
'step dpsi for the set number of steps
dpsi = ((((-hub_length / 2) - hub_leading_end(1, 1).y) / P) * 2 * pi) /
(num_edge_points - 1)
psi = psi - dpsi
'helical edge
'**********************
For i = 1 To num_edge_points
psi = psi + dpsi
temp(i).x = r * Cos(psi)
temp(i).y = (P * psi) / (2 * pi)
temp(i).z = -r * Sin(psi)
Next

Dim s As Single

'distance between points
s = Sqr((temp(1).x - final_section(1, section_data(1).le_id).x) ^ 2 + (temp(1).z -
final_section(1, section_data(1).le_id).z) ^ 2)
angle = 2 * asin((s / 2) / r)
Call rotate(temp(), angle, 100)

For i = 2 To num_edge_points
hub_leading_end(1, i).x = temp(i).x
hub_leading_end(1, i).y = temp(i).y
hub_leading_end(1, i).z = temp(i).z
Next


End Sub



Public Sub hub_nodes(node_count, hub_node_connect)
Dim i As Integer

'hub nodes
'**************************************************************
*****


'hub leading end
'nodes at front of hub

'first section is backwards
node_count = node_count + 1
hub_node_connect(1) = node_count
Print #1, "node", node_count, hub_leading_end(1, 100).x, hub_leading_end(1, 100).y,
hub_leading_end(1, 100).z

'rest of sections
For i = 2 To no_of_hub_strips + 1
node_count = node_count + 1
hub_node_connect(i) = node_count
Print #1, "node", node_count, hub_leading_end(i, 1).x, hub_leading_end(i, 1).y,
hub_leading_end(i, 1).z
Next


'hub leading v section between blades leading edges
'first is omitted since node defined with blade
'first node is 12
For i = 2 To no_of_hub_strips + 1
node_count = node_count + 1
hub_node_connect(10 + i) = node_count
Print #1, "node", node_count, hub_leading_end(i, 100).x, hub_leading_end(i, 100).y,
hub_leading_end(i, 100).z
Next

'hub trailing v section between blades trailing edges
'last node of section is omitted since node defined with blade
For i = 1 To no_of_hub_strips
node_count = node_count + 1
hub_node_connect(20 + 2 + no_of_hub_strips - i) = node_count
```

```
Print #1, "node", node_count, hub_trailing_end(i, 1).x, hub_trailing_end(i, 1).y,
hub_trailing_end(i, 1).z
Next


'hub trailing end
'nodes at end of hub
For i = 1 To no_of_hub_strips + 1
node_count = node_count + 1
hub_node_connect(30 + 2 + no_of_hub_strips - i) = node_count
Print #1, "node", node_count, hub_trailing_end(i, 100).x, hub_trailing_end(i, 100).y,
hub_trailing_end(i, 100).z
Next

End Sub
Public Sub ring_nodes(node_count, ring_node_connect)

Dim i As Integer

'ring nodes
'*******************************************************************
*****

'ring node on trailing edge of front blade
node_count = node_count + 1
ring_node_connect(1) = node_count
Print #1, "node", node_count, ring_section_edge(1).x, ring_section_edge(1).y,
ring_section_edge(1).z

'ring node on trailing edge of front blade
node_count = node_count + 1
ring_node_connect(2) = node_count
Print #1, "node", node_count, ring_section_edge(num_ring_section_points).x,
ring_section_edge(num_ring_section_points).y,
ring_section_edge(num_ring_section_points).z

End Sub

Public Sub duct_nodes(node_count, duct_node_connect)
'duct nodes
'*******************************************************************
*****
Dim j As Integer

For j = 1 To 10
'leading lower edge of duct
node_count = node_count + 1
duct_node_connect(j * 4 - 3) = node_count
Print #1, "node", node_count, duct_leading_lower(j, 1).x, duct_leading_lower(j, 1).y,
duct_leading_lower(j, 1).z
```

```
node_count = node_count + 1
duct_node_connect(j * 4 - 2) = node_count
Print #1, "node", node_count, duct_leading_lower(j, num_leading_lower_points).x,
duct_leading_lower(j, num_leading_lower_points).y, duct_leading_lower(j,
num_leading_lower_points).z

'trailing lower edge of duct
node_count = node_count + 1
duct_node_connect(j * 4 - 1) = node_count
Print #1, "node", node_count, duct_trailing_lower(j, 1).x, duct_trailing_lower(j, 1).y,
duct_trailing_lower(j, 1).z
node_count = node_count + 1
duct_node_connect(j * 4) = node_count
Print #1, "node", node_count, duct_trailing_lower(j, num_trailing_lower_points).x,
duct_trailing_lower(j, num_trailing_lower_points).y, duct_trailing_lower(j,
num_trailing_lower_points).z
Next

End Sub
Public Sub cap_nodes(node_count, cap_node_connect)
'cap nodes
'*****************************************************************
*****
Dim j As Integer

'node at nose of cap
node_count = node_count + 1
cap_node_connect(1) = node_count
Print #1, "node", node_count, cap_left(1).x, cap_left(1).y, cap_left(1).z

'node at corner near blade
node_count = node_count + 1
cap_node_connect(2) = node_count
Print #1, "node", node_count, cap_left(cap_section_points).x,
cap_left(cap_section_points).y, cap_left(cap_section_points).z

'node at other corner
node_count = node_count + 1
cap_node_connect(3) = node_count
Print #1, "node", node_count, cap_right(cap_section_points).x,
cap_right(cap_section_points).y, cap_right(cap_section_points).z

'node at middle of cap surface
node_count = node_count + 1
cap_node_connect(4) = node_count
Print #1, "node", node_count, cap_internal_x(1).x, cap_internal_x(1).y,
cap_internal_x(1).z

'node at mid of cap_left
node_count = node_count + 1
```

```
cap_node_connect(5) = node_count
Print #1, "node", node_count, cap_left(cap_sidenode_id).x,
cap_left(cap_sidenode_id).y, cap_left(cap_sidenode_id).z

'node at mid of cap_right
node_count = node_count + 1
cap_node_connect(6) = node_count
Print #1, "node", node_count, cap_right(cap_sidenode_id).x,
cap_right(cap_sidenode_id).y, cap_right(cap_sidenode_id).z

'node at middle of cap back
node_count = node_count + 1
cap_node_connect(7) = node_count
Print #1, "node", node_count, cap_internal_x(num_internal_x_points).x,
cap_internal_x(num_internal_x_points).y, cap_internal_x(num_internal_x_points).z


'this is not needed
'closes the cap to make a bullet
'*******************************************************************
*******
'node at corner near blade
'node_count = node_count + 1
'cap_node_connect(8) = node_count
'Print #1, "node", node_count, cap_left(cap_section_points).x, "0",
cap_left(cap_section_points).z

'node at middle of cap back
'node_count = node_count + 1
'cap_node_connect(9) = node_count
'Print #1, "node", node_count, cap_internal_x(num_internal_x_points).x, "0",
cap_internal_x(num_internal_x_points).z


'node at other corner
'node_count = node_count + 1
'cap_node_connect(10) = node_count
'Print #1, "node", node_count, cap_right(cap_section_points).x, "0",
cap_right(cap_section_points).z



End Sub
Public Sub second_cap_nodes(node_count, second_cap_node_connect)
'cap nodes
'*******************************************************************
*****
Dim j As Integer

'node at nose of cap
```

```
node_count = node_count + 1
second_cap_node_connect(1) = node_count
Print #1, "node", node_count, cap_left(1).x, -cap_left(1).y, cap_left(1).z

'node at corner near blade
node_count = node_count + 1
second_cap_node_connect(2) = node_count
Print #1, "node", node_count, cap_left(cap_section_points).x, -
cap_left(cap_section_points).y, cap_left(cap_section_points).z

'node at other corner
node_count = node_count + 1
second_cap_node_connect(3) = node_count
Print #1, "node", node_count, cap_right(cap_section_points).x, -
cap_right(cap_section_points).y, cap_right(cap_section_points).z

'node at middle of cap surface
node_count = node_count + 1
second_cap_node_connect(4) = node_count
Print #1, "node", node_count, cap_internal_x(1).x, -cap_internal_x(1).y,
cap_internal_x(1).z

'node at mid of cap_left
node_count = node_count + 1
second_cap_node_connect(5) = node_count
Print #1, "node", node_count, cap_internal_y(1).x, -cap_internal_y(1).y,
cap_internal_y(1).z

'node at mid of cap_right
node_count = node_count + 1
second_cap_node_connect(6) = node_count
Print #1, "node", node_count, cap_internal_z(1).x, -cap_internal_z(1).y,
cap_internal_z(1).z

'node at middle of cap surface
node_count = node_count + 1
second_cap_node_connect(7) = node_count
Print #1, "node", node_count, cap_internal_x(num_internal_x_points).x, -
cap_internal_x(num_internal_x_points).y, cap_internal_x(num_internal_x_points).z


End Sub

Public Sub duct_4ring_edges(edge_count, duct_node_connect, duct_edge_connect)
Dim i, j As Integer

'*****************************************************************
******
'duct edges
```

```
'**********************************************************************
******
For j = 1 To 10
'lower trailing edge
edge_count = edge_count + 1
duct_edge_connect(80 + j) = edge_count
Print #1, "edge", edge_count, "-4", num_leading_lower_points,
duct_leading_lower_panels_s, "1.0", "0.1"
Print #1, "startnode", duct_node_connect(4 * j - 3)
For i = 2 To num_leading_lower_points - 1
Print #1, (i - 1), duct_leading_lower(j, i).x, duct_leading_lower(j, i).y,
duct_leading_lower(j, i).z
Next
Print #1, "finishnode", duct_node_connect(4 * j - 2)

'lower leading edge
edge_count = edge_count + 1
duct_edge_connect(70 + j) = edge_count
Print #1, "edge", edge_count, "-4", num_trailing_lower_points,
duct_trailing_lower_panels_s, "1.0", "0.1"
Print #1, "startnode", duct_node_connect(4 * j - 1)
For i = 2 To num_trailing_lower_points - 1
Print #1, (i - 1), duct_trailing_lower(j, i).x, duct_trailing_lower(j, i).y,
duct_trailing_lower(j, i).z
Next
Print #1, "finishnode", duct_node_connect(4 * j)

'upper edge
edge_count = edge_count + 1
duct_edge_connect(j * 2) = edge_count
Print #1, "edge", edge_count, "-4", num_upper_points, duct_upper_panels_s, "1.0",
"0.1"
Print #1, "startnode", duct_node_connect(4 * j)
For i = 2 To num_upper_points - 1
Print #1, (i - 1), duct_upper(j, i).x, duct_upper(j, i).y, duct_upper(j, i).z
Next
Print #1, "finishnode", duct_node_connect(4 * j - 3)

Next j
For j = 1 To 9

'duct trailing arc joins to the wakesheet
edge_count = edge_count + 1
duct_edge_connect(20 + j * 2 - 1) = edge_count
Print #1, "edge", edge_count, "-3", "36", duct_panels_t, "1.0", "0.1"
Print #1, "startnode", duct_node_connect(4 * j - 3)
For i = 2 To 35
Print #1, (i - 1), duct_leading_arc(j, i).x, duct_leading_arc(j, i).y, duct_leading_arc(j,
i).z
Next
```

```
Print #1, "finishnode", duct_node_connect(4 * j + 1)

'duct leading arc
edge_count = edge_count + 1
duct_edge_connect(20 + j * 2) = edge_count
Print #1, "edge", edge_count, "-1", "36", duct_panels_t, "1.0", "0.1"
Print #1, "startnode", duct_node_connect(4 * j)
For i = 2 To 35
Print #1, (i - 1), duct_trailing_arc(j, i).x, duct_trailing_arc(j, i).y, duct_trailing_arc(j,
i).z
Next
Print #1, "finishnode", duct_node_connect(4 * (j + 1))

'duct lower trailing arc
edge_count = edge_count + 1
duct_edge_connect(60 + j) = edge_count
Print #1, "edge", edge_count, "-2", "36", duct_panels_t, "1.0", "0.1"
Print #1, "startnode", duct_node_connect(4 * j - 2)
For i = 2 To 35
Print #1, (i - 1), duct_leading_lower_arc(j, i).x, duct_leading_lower_arc(j, i).y,
duct_leading_lower_arc(j, i).z
Next
Print #1, "finishnode", duct_node_connect(4 * j + 2)

'duct lower leading arc
edge_count = edge_count + 1
duct_edge_connect(50 + j) = edge_count
Print #1, "edge", edge_count, "-2", "36", duct_panels_t, "1.0", "0.1"
Print #1, "startnode", duct_node_connect(4 * j - 1)
For i = 2 To 35
Print #1, (i - 1), duct_trailing_lower_arc(j, i).x, duct_trailing_lower_arc(j, i).y,
duct_trailing_lower_arc(j, i).z
Next
Print #1, "finishnode", duct_node_connect(4 * j + 3)

Next j

End Sub
Public Sub duct_no_ring_edges(edge_count, duct_node_connect, duct_edge_connect)
Dim i, j As Integer

'*****************************************************************
******
'duct edges
'for no ring duct
'*****************************************************************
******
For j = 1 To 10
'lower section
edge_count = edge_count + 1
```

```
duct_edge_connect(j * 2 - 1) = edge_count
If j = 1 Or j = 10 Then Print #1, "edge", edge_count, "-4", section_data(0).le_id,
duct_upper_panels_s, 2 - duct_P, duct_Q
If j <> 1 And j <> 10 Then Print #1, "edge", edge_count, "-1", section_data(0).le_id,
duct_upper_panels_s, 2 - duct_P, duct_Q
Print #1, "startnode", duct_node_connect(j * 4 - 3)
For i = 2 To section_data(0).le_id - 1
Print #1, (i - 1), duct_lower(j, i).x, duct_lower(j, i).y, duct_lower(j, i).z
Next
Print #1, "finishnode", duct_node_connect(j * 4)

'upper edge
edge_count = edge_count + 1
duct_edge_connect(j * 2) = edge_count
If j = 1 Or j = 10 Then Print #1, "edge", edge_count, "-4", num_upper_points,
duct_upper_panels_s, duct_P, duct_Q
If j <> 1 And j <> 10 Then Print #1, "edge", edge_count, "-1", num_upper_points,
duct_upper_panels_s, duct_P, duct_Q
Print #1, "startnode", duct_node_connect(j * 4)
For i = 2 To num_upper_points - 1
Print #1, (i - 1), duct_upper(j, i).x, duct_upper(j, i).y, duct_upper(j, i).z
Next
Print #1, "finishnode", duct_node_connect(j * 4 - 3)
Next

For j = 1 To 9
'duct trailing arc joins to the wakesheet
edge_count = edge_count + 1
duct_edge_connect(20 + j * 2 - 1) = edge_count
Print #1, "edge", edge_count, "-3", "36", duct_panels_t, "1.0", "0.1"
Print #1, "startnode", duct_node_connect(j * 4 - 3)
For i = 2 To 35
Print #1, (i - 1), duct_leading_arc(j, i).x, duct_leading_arc(j, i).y, duct_leading_arc(j,
i).z
Next
Print #1, "finishnode", duct_node_connect(j * 4 + 1)

'duct leading arc
edge_count = edge_count + 1
duct_edge_connect(20 + j * 2) = edge_count
Print #1, "edge", edge_count, "-1", "36", duct_panels_t, "1.0", "0.1"
Print #1, "startnode", duct_node_connect(j * 4)
For i = 2 To 35
Print #1, (i - 1), duct_trailing_arc(j, i).x, duct_trailing_arc(j, i).y, duct_trailing_arc(j,
i).z
Next
Print #1, "finishnode", duct_node_connect(j * 4 + 4)
Next
End Sub
Public Sub cap_faces(face_count, cap_node_connect, cap_edge_connect)
```

```
'nose face
face_count = face_count + 1
Print #1, "face", face_count, "16", "0"
Print #1, "linear"
Print #1, "origin node", cap_node_connect(1)
Print #1, "side", "0", "1", cap_edge_connect(1)
Print #1, "side", "1", "1", cap_edge_connect(8)
Print #1, "side", "2", "1", cap_edge_connect(9)
Print #1, "side", "3", "1", cap_edge_connect(3)
Print #1, "sources", "0", "0", "0"

'face next to cap_right
face_count = face_count + 1
Print #1, "face", face_count, "16", "0"
Print #1, "linear"
Print #1, "origin node", cap_node_connect(4)
Print #1, "side", "0", "1", cap_edge_connect(7)
Print #1, "side", "1", "1", cap_edge_connect(6)
Print #1, "side", "2", "1", cap_edge_connect(4)
Print #1, "side", "3", "1", cap_edge_connect(9)
Print #1, "sources", "0", "0", "0"
'face next to cap_left
face_count = face_count + 1
Print #1, "face", face_count, "16", "0"
Print #1, "linear"
Print #1, "origin node", cap_node_connect(5)
Print #1, "side", "0", "1", cap_edge_connect(2)
Print #1, "side", "1", "1", cap_edge_connect(5)
Print #1, "side", "2", "1", cap_edge_connect(7)
Print #1, "side", "3", "1", cap_edge_connect(8)
Print #1, "sources", "0", "0", "0"

'not needed
'creates cylinder for bullet
'*************************************************************
'face_count = face_count + 1
'Print #1, "face", face_count, "16", "0"
'Print #1, "linear"
'Print #1, "origin node", cap_node_connect(2)
'Print #1, "side", "0", "1", cap_edge_connect(12)
'Print #1, "side", "1", "1", cap_edge_connect(10)
'Print #1, "side", "2", "1", cap_edge_connect(13)
'Print #1, "side", "3", "1", cap_edge_connect(5)
'Print #1, "sources", "0", "0", "0"

'face_count = face_count + 1
'Print #1, "face", face_count, "16", "0"
'Print #1, "linear"
'Print #1, "origin node", cap_node_connect(7)
'Print #1, "side", "0", "1", cap_edge_connect(13)
```

```
'Print #1, "side", "1", "1", cap_edge_connect(11)
'Print #1, "side", "2", "1", cap_edge_connect(14)
'Print #1, "side", "3", "1", cap_edge_connect(6)
'Print #1, "sources", "0", "0", "0"

End Sub
Public Sub second_cap_faces(face_count, second_cap_node_connect,
second_cap_edge_connect)
'nose face
face_count = face_count + 1
Print #1, "face", face_count, "16", "0"
Print #1, "linear"
Print #1, "origin node", second_cap_node_connect(1)
Print #1, "side", "0", "1", second_cap_edge_connect(3)
Print #1, "side", "1", "1", second_cap_edge_connect(9)
Print #1, "side", "2", "1", second_cap_edge_connect(8)
Print #1, "side", "3", "1", second_cap_edge_connect(1)
Print #1, "sources", "0", "0", "0"

'face next to cap_right
face_count = face_count + 1
Print #1, "face", face_count, "16", "0"
Print #1, "linear"
Print #1, "origin node", second_cap_node_connect(4)
Print #1, "side", "0", "1", second_cap_edge_connect(9)
Print #1, "side", "1", "1", second_cap_edge_connect(4)
Print #1, "side", "2", "1", second_cap_edge_connect(6)
Print #1, "side", "3", "1", second_cap_edge_connect(7)
Print #1, "sources", "0", "0", "0"
'face next to cap_left
face_count = face_count + 1
Print #1, "face", face_count, "16", "0"
Print #1, "linear"
Print #1, "origin node", second_cap_node_connect(5)
Print #1, "side", "0", "1", second_cap_edge_connect(8)
Print #1, "side", "1", "1", second_cap_edge_connect(7)
Print #1, "side", "2", "1", second_cap_edge_connect(5)
Print #1, "side", "3", "1", second_cap_edge_connect(2)
Print #1, "sources", "0", "0", "0"
End Sub

Public Sub duct_4ring_faces(face_count, duct_node_connect, duct_edge_connect)
Dim j As Integer
'duct for ring
'has a gap on the lower side
'duct faces
'*******************************************************************
*******
For j = 1 To 9
'leading lower face
```

```
face_count = face_count + 1
Print #1, "face", face_count, "16", "3"
Print #1, "linear"
Print #1, "origin node", duct_node_connect(j * 4 + 4)
Print #1, "side", "0", "1", duct_edge_connect(71 + j)
Print #1, "side", "1", "1", duct_edge_connect(50 + j)
Print #1, "side", "2", "1", duct_edge_connect(70 + j)
Print #1, "side", "3", "1", duct_edge_connect(20 + j * 2)
Print #1, "sources", "0", "0", "0"

'upper face
face_count = face_count + 1
Print #1, "face", face_count, "16", "3"
Print #1, "linear"
Print #1, "origin node", duct_node_connect(j * 4 + 1)
Print #1, "side", "0", "1", duct_edge_connect(j * 2 + 2)
Print #1, "side", "1", "1", duct_edge_connect(20 + j * 2)
Print #1, "side", "2", "1", duct_edge_connect(j * 2)
Print #1, "side", "3", "1", duct_edge_connect(20 + j * 2 - 1)
Print #1, "sources", "0", "0", "0"

'trailing lower face
face_count = face_count + 1
Print #1, "face", face_count, "16", "3"
Print #1, "linear"
Print #1, "origin node", duct_node_connect(j * 4 + 2)
Print #1, "side", "0", "1", duct_edge_connect(81 + j)
Print #1, "side", "1", "1", duct_edge_connect(20 + j * 2 - 1)
Print #1, "side", "2", "1", duct_edge_connect(80 + j)
Print #1, "side", "3", "1", duct_edge_connect(60 + j)
Print #1, "sources", "0", "0", "0"

Next

End Sub
Public Sub duct_no_ring_faces(face_count, duct_node_connect, duct_edge_connect)
Dim j As Integer

'duct for no ring

'duct faces
'***********************************************************************
*******
'upper face
For j = 1 To 9
face_count = face_count + 1
Print #1, "face", face_count, "16", "3"
Print #1, "linear"
Print #1, "origin node", duct_node_connect(j * 4 + 1)
Print #1, "side", "0", "1", duct_edge_connect(j * 2 + 2)
Print #1, "side", "1", "1", duct_edge_connect(20 + j * 2)
Print #1, "side", "2", "1", duct_edge_connect(j * 2)
Print #1, "side", "3", "1", duct_edge_connect(20 + j * 2 - 1)
Print #1, "sources", "0", "0", "0"

'lower face
face_count = face_count + 1
Print #1, "face", face_count, "16", "3"
Print #1, "linear"
Print #1, "origin node", duct_node_connect(j * 4 + 4)
Print #1, "side", "0", "1", duct_edge_connect(j * 2 + 1)
Print #1, "side", "1", "1", duct_edge_connect(20 + j * 2 - 1)
Print #1, "side", "2", "1", duct_edge_connect(j * 2 - 1)
Print #1, "side", "3", "1", duct_edge_connect(20 + j * 2)
Print #1, "sources", "0", "0", "0"

Next

End Sub
Public Sub missing_section_for_hub(edge_count, blade_node_connect,
blade_ends_connect, blade_section_connect, blade_edge_connect, blade_panels_t,
blade_panels_s, hub_node_connect, ring_node_connect)
  Dim q As Integer
  Dim i As Integer
  Dim num_upper_section_nodes As Integer

  'upper side of section
  edge_count = edge_count + 1
  blade_section_connect(2) = edge_count
  num_upper_section_nodes = 1 + section_data(1).num_points - section_data(1).le_id
  Print #1, "edge", edge_count, "-1", num_upper_section_nodes, blade_panels_s,
blade_P, blade_Q
  Print #1, "startnode", blade_node_connect(1)
  q = section_data(1).le_id
 For i = 2 To num_upper_section_nodes - 1
   q = q + 1
   Print #1, (i - 1), final_section(1, q).x, final_section(1, q).y, final_section(1, q).z
 Next
  Print #1, "finishnode", blade_node_connect(2)

End Sub

Public Sub hub_edges(edge_count, hub_node_connect, hub_edge_connect,
blade_node_connect, blade_section_connect, hub_panels_t, hub_panels_s)
Dim i, j As Integer
Dim step, old_step, new_step
'hub edges
'*********************************************************************
*****
```

```
'leading edge arcs on front of hub
'*****************************************************************
*
step = 120 / no_of_hub_strips
old_step = 2 - step
new_step = 0

For j = 1 To no_of_hub_strips
old_step = old_step + step
new_step = new_step + step
edge_count = edge_count + 1
hub_edge_connect(j) = edge_count
Print #1, "edge", edge_count, "-2", step, hub_panels_s, "1.0", "0.1"
Print #1, "startnode", hub_node_connect(j)
For i = old_step To new_step - 1
Print #1, (i + 1 - old_step), hub_leading_edge(i).x, hub_leading_edge(i).y,
hub_leading_edge(i).z
Next
Print #1, "finishnode", hub_node_connect(j + 1)

Next j


'leading end of hub helixes
For j = 1 To no_of_hub_strips + 1
edge_count = edge_count + 1
hub_edge_connect(10 + j) = edge_count

If j <> (no_of_hub_strips + 1) And j <> 1 Then Print #1, "edge", edge_count, "-1",
"100", hub_panels_t, "1.0", "0.1"
If j = (no_of_hub_strips + 1) Or j = 1 Then Print #1, "edge", edge_count, "-4", "100",
hub_panels_t, "1.0", "0.1"
If j = 1 Then Print #1, "startnode", blade_node_connect(1)
If j <> 1 Then Print #1, "startnode", hub_node_connect(j)
For i = 2 To 99
Print #1, (i - 1), hub_leading_end(j, i).x, hub_leading_end(j, i).y, hub_leading_end(j,
i).z
Next
If j = 1 Then Print #1, "finishnode", hub_node_connect(1)
If j <> 1 Then Print #1, "finishnode", hub_node_connect(10 + j)
Next


'leading V section
'*****************************************************************
*
step = 120 / no_of_hub_strips
old_step = 2 - step
new_step = 0
```

```
For j = 1 To no_of_hub_strips
old_step = old_step + step
new_step = new_step + step
edge_count = edge_count + 1
hub_edge_connect(20 + j) = edge_count
Print #1, "edge", edge_count, "-1", step, hub_panels_s, "1.0", "0.1"
If j = 1 Then Print #1, "startnode", blade_node_connect(1)
If j <> 1 Then Print #1, "startnode", hub_node_connect(10 + j)
For i = old_step To new_step - 1
Print #1, (i + 1 - old_step), hub_helix_le(i).x, hub_helix_le(i).y, hub_helix_le(i).z
Next
Print #1, "finishnode", hub_node_connect(10 + j + 1)
Next j


'helixes between blades
'*****************************************************************
*
For j = 2 To no_of_hub_strips
edge_count = edge_count + 1
hub_edge_connect(30 + j) = edge_count

Print #1, "edge", edge_count, "-1", "100", blade_panels_s, blade_P, blade_Q
Print #1, "startnode", hub_node_connect(10 + j)
For i = 2 To 99
Print #1, (i - 1), hub_helix_blades(j, i).x, hub_helix_blades(j, i).y, hub_helix_blades(j,
i).z
Next
Print #1, "finishnode", hub_node_connect(20 + j)
Next

'second hub blade
'not defined with blade
edge_count = edge_count + 1
hub_edge_connect(30 + 1 + no_of_hub_strips) = edge_count
Print #1, "edge", edge_count, "-4", section_data(1).le_id, blade_panels_s, 2 - blade_P,
blade_Q
Print #1, "startnode", hub_node_connect(20 + 1 + no_of_hub_strips) 'trailing edge
For i = 2 To section_data(1).le_id - 1
Print #1, (i - 1), hub_section_edge(10, i).x, hub_section_edge(10, i).y,
hub_section_edge(10, i).z
Next
Print #1, "finishnode", hub_node_connect(10 + 1 + no_of_hub_strips)

'leading V section
'*****************************************************************
*
step = 120 / no_of_hub_strips
old_step = 2 - step
new_step = 0
```

```
For j = 1 To no_of_hub_strips
old_step = old_step + step
new_step = new_step + step
edge_count = edge_count + 1
hub_edge_connect(40 + no_of_hub_strips + 1 - j) = edge_count
Print #1, "edge", edge_count, "-1", step, hub_panels_s, "1.0", "0.1"

Print #1, "startnode", hub_node_connect(20 + no_of_hub_strips + 2 - j)
For i = old_step To new_step - 1
Print #1, (i + 1 - old_step), hub_helix_te(i).x, hub_helix_te(i).y, hub_helix_te(i).z
Next
If j = no_of_hub_strips Then Print #1, "finishnode", blade_node_connect(2)
If j <> no_of_hub_strips Then Print #1, "finishnode", hub_node_connect(20 +
no_of_hub_strips + 1 - j)
Next j


'trailing end of hub helixes
For j = 1 To no_of_hub_strips + 1
edge_count = edge_count + 1
hub_edge_connect(50 + j) = edge_count

If j <> (no_of_hub_strips + 1) And j <> 1 Then Print #1, "edge", edge_count, "-1",
"100", hub_panels_t, "1.0", "0.1"
If j = (no_of_hub_strips + 1) Or j = 1 Then Print #1, "edge", edge_count, "-4", "100",
hub_panels_t, "1.0", "0.1"
If j = 1 Then Print #1, "startnode", blade_node_connect(2)
If j <> 1 Then Print #1, "startnode", hub_node_connect(20 + j)
For i = 2 To 99
Print #1, (i - 1), hub_trailing_end(no_of_hub_strips + 2 - j, i).x,
hub_trailing_end(no_of_hub_strips + 2 - j, i).y, hub_trailing_end(no_of_hub_strips +
2 - j, i).z
Next
Print #1, "finishnode", hub_node_connect(30 + j)
Next


'trailing edge arcs on front of hub
'*****************************************************************
*
step = 120 / no_of_hub_strips
old_step = 2 - step
new_step = 0

For j = 1 To no_of_hub_strips
old_step = old_step + step
new_step = new_step + step
edge_count = edge_count + 1
hub_edge_connect(60 + j) = edge_count
```

```
Print #1, "edge", edge_count, "-2", step, hub_panels_s, "1.0", "0.1"
Print #1, "startnode", hub_node_connect(30 + j)
For i = old_step To new_step - 1
Print #1, (i + 1 - old_step), hub_trailing_edge(i).x, hub_trailing_edge(i).y,
hub_trailing_edge(i).z
Next
Print #1, "finishnode", hub_node_connect(30 + j + 1)

Next j


End Sub
Public Sub ring_edges(edge_count, ring_node_connect, ring_edge_connect,
blade_node_connect, ring_panels_t)
Dim i, j As Integer

'ring edges
'*********************************************************************
*****

'leading edge
edge_count = edge_count + 1
ring_edge_connect(1) = edge_count
Print #1, "edge", edge_count, "-2", 121, ring_panels_t, "1.0", "0.1"
Print #1, "startnode", blade_node_connect(num_sections * 2 - 1)
For i = 2 To 120
Print #1, (i - 1), ring_leading_edge(i).x, ring_leading_edge(i).y,
ring_leading_edge(i).z
Next
Print #1, "finishnode", ring_node_connect(2)

'trailing edge
edge_count = edge_count + 1
ring_edge_connect(2) = edge_count
Print #1, "edge", edge_count, "-2", 121, ring_panels_t, "1.0", "0.1"
Print #1, "startnode", blade_node_connect(num_sections * 2)
For i = 2 To 120
Print #1, (i - 1), ring_trailing_edge(i).x, ring_trailing_edge(i).y,
ring_trailing_edge(i).z
Next
Print #1, "finishnode", ring_node_connect(1)

'second ring section edge
edge_count = edge_count + 1
ring_edge_connect(3) = edge_count
Print #1, "edge", edge_count, "-1", section_data(1).le_id, blade_panels_s, "1.0", "0.1"
Print #1, "startnode", ring_node_connect(1) 'trailing edge
For i = 2 To num_ring_section_points - 1
Print #1, (i - 1), ring_section_edge(i).x, ring_section_edge(i).y, ring_section_edge(i).z
Next
```

```
Print #1, "finishnode", ring_node_connect(2)

End Sub

Public Sub cap_edges(edge_count, cap_node_connect, cap_edge_connect,
cap_panels_t, cap_panels_s)
Dim i, j As Integer
Dim lower_points As Integer
Dim upper_points As Integer

upper_points = cap_section_points - cap_sidenode_id + 1
lower_points = cap_sidenode_id
'cap edges
'**********************************************************************
*****

'arc on left
'first part
edge_count = edge_count + 1
cap_edge_connect(1) = edge_count
Print #1, "edge", edge_count, "-4", lower_points, cap_panels_t, "1.0", "0.1"
Print #1, "startnode", cap_node_connect(1)
For i = 2 To cap_sidenode_id - 1
Print #1, (i - 1), cap_left(i).x, cap_left(i).y, cap_left(i).z
Next
Print #1, "finishnode", cap_node_connect(5)

'arc on left
'second part
edge_count = edge_count + 1
cap_edge_connect(2) = edge_count
Print #1, "edge", edge_count, "-4", upper_points, cap_panels_s, "1.0", "0.1"
Print #1, "startnode", cap_node_connect(5)
For i = cap_sidenode_id + 1 To cap_section_points - 1
Print #1, (i - cap_sidenode_id), cap_left(i).x, cap_left(i).y, cap_left(i).z
Next
Print #1, "finishnode", cap_node_connect(2)


'arc on right
'first part
edge_count = edge_count + 1
cap_edge_connect(3) = edge_count
Print #1, "edge", edge_count, "-4", lower_points, cap_panels_t, "1.0", "0.1"
Print #1, "startnode", cap_node_connect(1)
For i = 2 To cap_sidenode_id - 1
Print #1, (i - 1), cap_right(i).x, cap_right(i).y, cap_right(i).z
Next
Print #1, "finishnode", cap_node_connect(6)
```

```
'arc on right
'second part
edge_count = edge_count + 1
cap_edge_connect(4) = edge_count
Print #1, "edge", edge_count, "-4", upper_points, cap_panels_s, "1.0", "0.1"
Print #1, "startnode", cap_node_connect(6)
For i = cap_sidenode_id + 1 To cap_section_points - 1
Print #1, (i - cap_sidenode_id), cap_right(i).x, cap_right(i).y, cap_right(i).z
Next
Print #1, "finishnode", cap_node_connect(3)


'arc on back
'first part
edge_count = edge_count + 1
cap_edge_connect(5) = edge_count
Print #1, "edge", edge_count, "-2", "19", cap_panels_t, "1.0", "0.1"
Print #1, "startnode", cap_node_connect(2)
For i = 2 To 18
Print #1, (i - 1), cap_back(i).x, cap_back(i).y, cap_back(i).z
Next
Print #1, "finishnode", cap_node_connect(7)

'arc on back
'second part
edge_count = edge_count + 1
cap_edge_connect(6) = edge_count
Print #1, "edge", edge_count, "-2", "19", cap_panels_t, "1.0", "0.1"
Print #1, "startnode", cap_node_connect(7)
For i = 19 To 35
Print #1, (i - 18), cap_back(i).x, cap_back(i).y, cap_back(i).z
Next
Print #1, "finishnode", cap_node_connect(3)


'arc internal_x
edge_count = edge_count + 1
cap_edge_connect(7) = edge_count
Print #1, "edge", edge_count, "-1", num_internal_x_points, cap_panels_s, "1.0", "0.1"
Print #1, "startnode", cap_node_connect(4)
For i = 2 To num_internal_x_points - 1
Print #1, (i - 1), cap_internal_x(i).x, cap_internal_x(i).y, cap_internal_x(i).z
Next
Print #1, "finishnode", cap_node_connect(7)

'arc internal_y
edge_count = edge_count + 1
cap_edge_connect(8) = edge_count
Print #1, "edge", edge_count, "-1", num_internal_points + 2, cap_panels_t, "1.0",
"0.1"
```

```
Print #1, "startnode", cap_node_connect(5)
For i = 1 To num_internal_points
Print #1, (i), cap_internal_y(i).x, cap_internal_y(i).y, cap_internal_y(i).z
Next
Print #1, "finishnode", cap_node_connect(4)

'arc internal_z
edge_count = edge_count + 1
cap_edge_connect(9) = edge_count
Print #1, "edge", edge_count, "-1", num_internal_points + 2, cap_panels_t, "1.0",
"0.1"
Print #1, "startnode", cap_node_connect(6)
For i = 1 To num_internal_points
Print #1, (i), cap_internal_z(i).x, cap_internal_z(i).y, cap_internal_z(i).z
Next
Print #1, "finishnode", cap_node_connect(4)

'not needed
'creates bullet
'*****************************************************************
'arc on back
'first part
'edge_count = edge_count + 1
'cap_edge_connect(10) = edge_count
'Print #1, "edge", edge_count, "-2", "19", cap_panels_t, "1.0", "0.1"
'Print #1, "startnode", cap_node_connect(8)
'For i = 2 To 18
'Print #1, (i - 1), cap_back(i).x, "0", cap_back(i).z
'Next
'Print #1, "finishnode", cap_node_connect(9)

'arc on back
'second part
'edge_count = edge_count + 1
'cap_edge_connect(11) = edge_count
'Print #1, "edge", edge_count, "-2", "19", cap_panels_t, "1.0", "0.1"
'Print #1, "startnode", cap_node_connect(9)
'For i = 19 To 35
'Print #1, (i - 18), cap_back(i).x, "0", cap_back(i).z
'Next
'Print #1, "finishnode", cap_node_connect(10)

'edge_count = edge_count + 1
'cap_edge_connect(12) = edge_count
'Print #1, "edge", edge_count, "-4", "2", cap_panels_t, "1.0", "0.1"
'Print #1, "startnode", cap_node_connect(2)
'Print #1, "finishnode", cap_node_connect(8)'

'edge_count = edge_count + 1
'cap_edge_connect(13) = edge_count
```

```
'Print #1, "edge", edge_count, "-1", "2", cap_panels_t, "1.0", "0.1"
'Print #1, "startnode", cap_node_connect(7)
'Print #1, "finishnode", cap_node_connect(9)

'edge_count = edge_count + 1
'cap_edge_connect(14) = edge_count
'Print #1, "edge", edge_count, "-4", "2", cap_panels_t, "1.0", "0.1"
'Print #1, "startnode", cap_node_connect(3)
'Print #1, "finishnode", cap_node_connect(10)



End Sub
Public Sub second_cap_edges(edge_count, second_cap_node_connect,
second_cap_edge_connect, cap_panels_t, cap_panels_s)
Dim i, j As Integer
Dim lower_points As Integer
Dim upper_points As Integer

upper_points = cap_section_points - cap_sidenode_id + 1
lower_points = cap_sidenode_id
'cap edges
'*****************************************************************
*****

'arc on left
'first part
edge_count = edge_count + 1
second_cap_edge_connect(1) = edge_count
Print #1, "edge", edge_count, "-4", lower_points, cap_panels_t, "1.0", "0.1"
Print #1, "startnode", second_cap_node_connect(1)
For i = 2 To cap_sidenode_id - 1
Print #1, (i - 1), cap_left(i).x, -cap_left(i).y, cap_left(i).z
Next
Print #1, "finishnode", second_cap_node_connect(5)

'arc on left
'second part
edge_count = edge_count + 1
second_cap_edge_connect(2) = edge_count
Print #1, "edge", edge_count, "-4", upper_points, cap_panels_s, "1.0", "0.1"
Print #1, "startnode", second_cap_node_connect(5)
For i = cap_sidenode_id + 1 To cap_section_points - 1
Print #1, (i - cap_sidenode_id), cap_left(i).x, -cap_left(i).y, cap_left(i).z
Next
Print #1, "finishnode", second_cap_node_connect(2)


'arc on right
'first part
```

```
edge_count = edge_count + 1
second_cap_edge_connect(3) = edge_count
Print #1, "edge", edge_count, "-4", lower_points, cap_panels_t, "1.0", "0.1"
Print #1, "startnode", second_cap_node_connect(1)
For i = 2 To cap_sidenode_id - 1
Print #1, (i - 1), cap_right(i).x, -cap_right(i).y, cap_right(i).z
Next
Print #1, "finishnode", second_cap_node_connect(6)

'arc on right
'second part
edge_count = edge_count + 1
second_cap_edge_connect(4) = edge_count
Print #1, "edge", edge_count, "-4", upper_points, cap_panels_s, "1.0", "0.1"
Print #1, "startnode", second_cap_node_connect(6)
For i = cap_sidenode_id + 1 To cap_section_points - 1
Print #1, (i - cap_sidenode_id), cap_right(i).x, -cap_right(i).y, cap_right(i).z
Next
Print #1, "finishnode", second_cap_node_connect(3)


'arc on back
'first part
edge_count = edge_count + 1
second_cap_edge_connect(5) = edge_count
Print #1, "edge", edge_count, "-2", "19", cap_panels_t, "1.0", "0.1"
Print #1, "startnode", second_cap_node_connect(2)
For i = 2 To 18
Print #1, (i - 1), cap_back(i).x, -cap_back(i).y, cap_back(i).z
Next
Print #1, "finishnode", second_cap_node_connect(7)

'arc on back
'second part
edge_count = edge_count + 1
second_cap_edge_connect(6) = edge_count
Print #1, "edge", edge_count, "-2", "19", cap_panels_t, "1.0", "0.1"
Print #1, "startnode", second_cap_node_connect(7)
For i = 19 To 35
Print #1, (i - 18), cap_back(i).x, -cap_back(i).y, cap_back(i).z
Next
Print #1, "finishnode", second_cap_node_connect(3)


'arc internal_x
edge_count = edge_count + 1
second_cap_edge_connect(7) = edge_count
Print #1, "edge", edge_count, "-1", num_internal_x_points, cap_panels_s, "1.0", "0.1"
Print #1, "startnode", second_cap_node_connect(4)
For i = 2 To num_internal_x_points - 1
```

```
Print #1, (i - 1), cap_internal_x(i).x, -cap_internal_x(i).y, cap_internal_x(i).z
Next
Print #1, "finishnode", second_cap_node_connect(7)

'arc internal_y
edge_count = edge_count + 1
second_cap_edge_connect(8) = edge_count
Print #1, "edge", edge_count, "-1", num_internal_points + 2, cap_panels_t, "1.0",
"0.1"
Print #1, "startnode", second_cap_node_connect(5)
For i = 1 To num_internal_points
Print #1, (i), cap_internal_y(i).x, -cap_internal_y(i).y, cap_internal_y(i).z
Next
Print #1, "finishnode", second_cap_node_connect(4)

'arc internal_z
edge_count = edge_count + 1
second_cap_edge_connect(9) = edge_count
Print #1, "edge", edge_count, "-1", num_internal_points + 2, cap_panels_t, "1.0",
"0.1"
Print #1, "startnode", second_cap_node_connect(6)
For i = 1 To num_internal_points
Print #1, (i), cap_internal_z(i).x, -cap_internal_z(i).y, cap_internal_z(i).z
Next
Print #1, "finishnode", second_cap_node_connect(4)


End Sub

Public Sub output_flx()
Dim node_count, i, face_count As Integer
Dim edge_count, duct_panels_s, duct_panels_t As Integer
'Dim ring_panels_s, ring_panels_t As Integer
'Dim hub_panels_s, hub_panels_t As Integer
'Dim blade_panels_s, blade_panels_t As Integer
'Dim duct_wake_panels_s, duct_wake_panels_t As Integer
'Dim blade_wake_panels_s, blade_wake_panels_t As Integer
Dim t_nodes, t_edges, t_faces As Integer
Dim num_cap_nodes, num_cap_edges, num_cap_faces As Integer
Dim num_ring_nodes, num_duct_nodes, num_hub_nodes, num_blade_nodes,
num_duct_wake_nodes, num_blade_wake_nodes As Integer
Dim num_duct_edges, num_ring_edges, num_hub_edges, num_blade_edges,
num_duct_wake_edges, num_blade_wake_edges As Integer
Dim num_duct_faces, num_ring_faces, num_hub_faces, num_blade_faces,
num_duct_wake_faces, num_blade_wake_faces As Integer



Dim cap_node_connect(7) As Integer 'should be 7
Dim cap_edge_connect(9) As Integer 'should be 9
Dim second_cap_node_connect(7) As Integer
Dim second_cap_edge_connect(9) As Integer

Dim duct_node_connect(60) As Integer
Dim duct_edge_connect(90) As Integer
Dim ring_node_connect(60) As Integer
Dim ring_edge_connect(90) As Integer
Dim hub_node_connect(60) As Integer
Dim hub_edge_connect(90) As Integer
Dim blade_ends_connect(6) As Integer
Dim duct_wake_node_connect(20) As Integer
Dim duct_wake_edge_connect(38) As Integer


Dim blade_node_connect() As Integer
Dim blade_edge_connect() As Integer
Dim blade_section_connect() As Integer
Dim blade_wake_vertedge_connect() As Integer
Dim blade_wake_secondedge_connect() As Integer

ReDim blade_node_connect(num_sections * 2)
ReDim blade_edge_connect((num_sections - 1) * 2)
ReDim blade_section_connect(num_sections * 2)

ReDim blade_wake_node_connect(num_sections * 2)
ReDim blade_wake_edge_connect(num_sections)
ReDim blade_wake_secondedge_connect(num_sections)
ReDim blade_wake_vertedge_connect(2 * (num_sections - 1))

node_count = -1
edge_count = -1
face_count = -1



num_cap_nodes = 7
num_ring_nodes = 2
num_duct_nodes = 40
num_hub_nodes = 4 * (no_of_hub_strips + 1) - 2
num_duct_wake_nodes = 20
num_blade_nodes = num_sections * 2
num_blade_wake_nodes = num_sections * 2

num_ring_edges = no_of_ring_strips * 9 + 4
If ring_true = 1 Then
num_duct_edges = 66
Else
num_duct_edges = 38
End If
num_ring_edges = 3
```

```
num_cap_edges = 9
num_hub_edges = no_of_hub_strips * 7 + 2
num_duct_wake_edges = 38
num_blade_edges = num_sections * 2 + (num_sections - 1) * 2
num_blade_wake_edges = num_sections * 2 + (num_sections - 1) * 2

If ring_true = 1 Then
num_duct_faces = 27
num_ring_faces = 1
Else
num_ring_faces = 0
num_duct_faces = 18
End If
num_cap_faces = 3
num_hub_faces = no_of_hub_strips * 3
num_duct_wake_faces = 18
num_blade_faces = (num_sections - 1) * 2
num_blade_wake_faces = (num_sections - 1) * 2

t_nodes = 0
t_edges = 0
t_faces = 0

If frmMain.ring_flag.value = 1 Then
t_nodes = t_nodes + num_ring_nodes
t_edges = t_edges + num_ring_edges
t_faces = t_faces + num_ring_faces
End If
If frmMain.hub_flag.value = 1 Then
t_nodes = t_nodes + num_hub_nodes
t_edges = t_edges + num_hub_edges
t_faces = t_faces + num_hub_faces
   If frmMain.blade_flag.value = 0 Then
   t_nodes = t_nodes + 2
   t_edges = t_edges + 1
   End If
End If
If frmMain.cap_flag.value = 1 Then
t_nodes = t_nodes + num_cap_nodes * 2
t_edges = t_edges + num_cap_edges * 2
t_faces = t_faces + num_cap_faces * 2
End If
If frmMain.blade_flag.value = 1 Then
t_nodes = t_nodes + num_blade_nodes + num_blade_wake_nodes
t_edges = t_edges + num_blade_edges + num_blade_wake_edges
t_faces = t_faces + num_blade_faces + num_blade_wake_faces
End If

If frmMain.duct_flag.value = 1 Then
t_nodes = t_nodes + num_duct_nodes + num_duct_wake_nodes
```

```
t_edges = t_edges + num_duct_edges + num_duct_wake_edges
t_faces = t_faces + num_duct_faces + num_duct_wake_faces
End If



Open "out.flx" For Output As 1
Print #1, "Thruster"
Print #1, "GRID TYPE PANEL"
Print #1, t_nodes, t_edges, t_faces, "0"

'node definition
'**************************************************
If frmMain.cap_flag.value = 1 Then
Call cap_nodes(node_count, cap_node_connect)
Call second_cap_nodes(node_count, second_cap_node_connect)
End If

If frmMain.duct_flag.value = 1 Then
Call duct_nodes(node_count, duct_node_connect)
Call duct_wake_nodes(node_count, duct_wake_node_connect)
End If
If frmMain.ring_flag.value = 1 Then Call ring_nodes(node_count,
ring_node_connect)
If frmMain.hub_flag.value = 1 Then
If frmMain.blade_flag.value = 0 Then Call missing_hub_nodes(node_count,
blade_node_connect)
Call hub_nodes(node_count, hub_node_connect)
End If
If frmMain.blade_flag.value = 1 Then
Call blade_nodes(node_count, blade_node_connect)
Call blade_wake_nodes(node_count, blade_wake_node_connect)
End If



'edge definition
'*****************************************************************
**********
If frmMain.duct_flag.value = 1 Then
If ring_true = 1 Then
  Call duct_4ring_edges(edge_count, duct_node_connect, duct_edge_connect)
Else
  Call duct_no_ring_edges(edge_count, duct_node_connect, duct_edge_connect)
End If
Call duct_wake_edges(edge_count, duct_wake_node_connect,
duct_wake_edge_connect, duct_node_connect)
End If
If frmMain.cap_flag.value = 1 Then
Call cap_edges(edge_count, cap_node_connect, cap_edge_connect, cap_panels_t,
cap_panels_s)
```

```
Call second_cap_edges(edge_count, second_cap_node_connect,
second_cap_edge_connect, cap_panels_t, cap_panels_s)
End If
If frmMain.ring_flag.value = 1 Then Call ring_edges(edge_count, ring_node_connect,
ring_edge_connect, blade_node_connect, ring_panels_t)

If frmMain.hub_flag.value = 1 Then
If frmMain.blade_flag.value = 0 Then Call missing_section_for_hub(edge_count,
blade_node_connect, blade_ends_connect, blade_section_connect,
blade_edge_connect, blade_panels_t, blade_panels_s, hub_node_connect,
ring_node_connect)
Call hub_edges(edge_count, hub_node_connect, hub_edge_connect,
blade_node_connect, blade_section_connect, hub_panels_t, hub_panels_s)
End If
If frmMain.blade_flag.value = 1 Then
Call blade_edges(edge_count, blade_node_connect, blade_ends_connect,
blade_section_connect, blade_edge_connect, blade_panels_t, blade_panels_s,
hub_node_connect, ring_node_connect)
Call blade_wake_edges(edge_count, blade_wake_node_connect,
blade_node_connect, blade_wake_edge_connect, hub_node_connect,
ring_node_connect, blade_wake_vertedge_connect,
blade_wake_secondedge_connect)
End If

'face definition
'*******************************************************************
**********
If frmMain.duct_flag.value = 1 Then
If ring_true = 1 Then
Call duct_4ring_faces(face_count, duct_node_connect, duct_edge_connect)
Else
Call duct_no_ring_faces(face_count, duct_node_connect, duct_edge_connect)
End If
Call duct_wake_faces(face_count, duct_wake_node_connect,
duct_wake_edge_connect, duct_edge_connect, duct_node_connect)
End If
If frmMain.ring_flag.value = 1 Then Call ring_faces(face_count, ring_node_connect,
ring_edge_connect, blade_section_connect)
If frmMain.cap_flag.value = 1 Then
Call cap_faces(face_count, cap_node_connect, cap_edge_connect)
Call second_cap_faces(face_count, second_cap_node_connect,
second_cap_edge_connect)
End If

If frmMain.hub_flag.value = 1 Then Call hub_faces(face_count, hub_node_connect,
hub_edge_connect, blade_section_connect, blade_edge_connect)

If frmMain.blade_flag.value = 1 Then
```

```
Call blade_faces(face_count, blade_node_connect, blade_ends_connect,
blade_section_connect, blade_edge_connect, hub_edge_connect, ring_edge_connect,
hub_node_connect)
Call blade_wake_faces(face_count, blade_wake_node_connect,
blade_wake_edge_connect, blade_edge_connect, blade_wake_vertedge_connect,
blade_ends_connect, blade_wake_secondedge_connect, hub_node_connect,
blade_node_connect)
End If

Close #1
End Sub
Public Sub blade_wake_faces(face_count, blade_wake_node_connect,
blade_wake_edge_connect, blade_edge_connect, blade_wake_vertedge_connect,
blade_ends_connect, blade_wake_secondedge_connect, hub_node_connect,
blade_node_connect)
Dim i As Integer

'faces for adapted wake
'**********************************************************************
****

For i = 1 To num_sections - 1
face_count = face_count + 1
Print #1, "face", face_count, "2048", "1"
Print #1, "linear"
Print #1, "origin node", blade_node_connect((i) * 2)
Print #1, "side", "0", "1", blade_wake_edge_connect(i)
Print #1, "side", "1", "1", blade_wake_vertedge_connect(i)
Print #1, "side", "2", "1", blade_wake_edge_connect(i + 1)
Print #1, "side", "3", "1", blade_edge_connect((i) * 2)
Print #1, "sources", "0", "0", "0"
Next

'faces for fixed wake
'****************************************************************
For i = 1 To num_sections - 1
face_count = face_count + 1
Print #1, "face", face_count, "4096", "1"
Print #1, "linear"
Print #1, "origin node", blade_wake_node_connect((i - 1) * 2 + 1)
Print #1, "side", "0", "1", blade_wake_secondedge_connect(i)
Print #1, "side", "1", "1", blade_wake_vertedge_connect((num_sections - 1) + i)
Print #1, "side", "2", "1", blade_wake_secondedge_connect(i + 1)
Print #1, "side", "3", "1", blade_wake_vertedge_connect(i)
Print #1, "sources", "0", "0", "0"
Next

End Sub
Public Sub blade_wake_edges(edge_count, blade_wake_node_connect,
blade_node_connect, blade_wake_edge_connect, hub_node_connect,
ring_node_connect, blade_wake_vertedge_connect,
blade_wake_secondedge_connect)
Dim j, i As Integer
Dim adapt, non_adapt As Integer

'free wake to 1/3 length
adapt = Int(no_of_wake_points * blade_freewake_length)

For j = 1 To num_sections
 edge_count = edge_count + 1
 blade_wake_edge_connect(j) = edge_count
 If j = 1 Or j = num_sections Then
 Print #1, "edge", edge_count, "-2", adapt, blade_freewake_panels_s, "0.3", "2.0"
 Print #1, "startnode", blade_node_connect((j - 1) * 2 + 2)
 End If
 If j <> 1 And j <> num_sections Then
 Print #1, "edge", edge_count, "-1", adapt, blade_freewake_panels_s, "0.3", "2.0"
 Print #1, "startnode", blade_node_connect((j - 1) * 2 + 2)
 End If
For i = 2 To adapt - 1
 Print #1, (i - 1), wake(j, i).x, wake(j, i).y, wake(j, i).z
Next
 Print #1, "finishnode", blade_wake_node_connect((j - 1) * 2 + 1)
Next

'edges perpendicular to the adapted wake sections
For j = 1 To num_sections - 1
 edge_count = edge_count + 1
 blade_wake_vertedge_connect(j) = edge_count
 If j = (num_sections - 1) Then Print #1, "edge", edge_count, "-1", "2",
Int(blade_panels_t * blade_tip_cluster), "1.0", "0.1"
 If j <> (num_sections - 1) Then Print #1, "edge", edge_count, "-1", "2",
blade_panels_t, "1.0", "0.1"
 Print #1, "startnode", blade_wake_node_connect((j - 1) * 2 + 1)
 Print #1, "finishnode", blade_wake_node_connect((j - 1) * 2 + 3)
Next

'fixed wake behind adapted wake
'*********************************************************************
*******
non_adapt = 1 + no_of_wake_points - adapt

For j = 1 To num_sections
 edge_count = edge_count + 1
 blade_wake_secondedge_connect(j) = edge_count
 If j = 1 Or j = num_sections Then Print #1, "edge", edge_count, "-2", non_adapt,
blade_fixedwake_panels_s, "1.0", "0.1"
 If j <> 1 And j <> num_sections Then Print #1, "edge", edge_count, "-1", non_adapt,
blade_fixedwake_panels_s, "1.0", "0.1"
 Print #1, "startnode", blade_wake_node_connect((j - 1) * 2 + 1)
```

```
For i = (adapt + 1) To no_of_wake_points - 1
 Print #1, (i - adapt), wake(j, i).x, wake(j, i).y, wake(j, i).z
Next
 Print #1, "finishnode", blade_wake_node_connect((j - 1) * 2 + 2)
Next

'edges perpendicular to the fixed wake sections (end of wake)
For j = 1 To num_sections - 1
 edge_count = edge_count + 1
 blade_wake_vertedge_connect((num_sections - 1) + j) = edge_count
 If j = (num_sections - 1) Then Print #1, "edge", edge_count, "-2", "2",
Int(blade_panels_t * blade_tip_cluster), "1.0", "0.1"
 If j <> (num_sections - 1) Then Print #1, "edge", edge_count, "-2", "2",
blade_panels_t, "1.0", "0.1"
 Print #1, "startnode", blade_wake_node_connect((j - 1) * 2 + 2)
 Print #1, "finishnode", blade_wake_node_connect((j - 1) * 2 + 4)
Next

End Sub
Public Sub blade_wake_nodes(node_count, blade_wake_node_connect)
'blade wake nodes
'**************************************************************
'only two nodes per section one at the adaption amd one at the end

Dim i As Integer
Dim adapt As Integer

adapt = Int(no_of_wake_points * blade_freewake_length)
For i = 1 To num_sections

'adaption node
node_count = node_count + 1
blade_wake_node_connect(((i - 1) * 2) + 1) = node_count
Print #1, "node", node_count, wake(i, adapt).x, wake(i, adapt).y, wake(i, adapt).z

'end node
node_count = node_count + 1
blade_wake_node_connect(((i - 1) * 2) + 2) = node_count
Print #1, "node", node_count, wake(i, no_of_wake_points).x, wake(i,
no_of_wake_points).y, wake(i, no_of_wake_points).z

Next
End Sub
Public Sub duct_wake_faces(face_count, duct_wake_node_connect,
duct_wake_edge_connect, duct_edge_connect, duct_node_connect)
Dim j As Integer

'duct wake sheet face
'************************************************
For j = 1 To 9
```

```
'free wake
face_count = face_count + 1
Print #1, "face", face_count, "2048", "3"
Print #1, "linear"
Print #1, "origin node", duct_node_connect(j * 4 + 1)
Print #1, "side", "0", "1", duct_wake_edge_connect(j * 2 + 1)
Print #1, "side", "1", "1", duct_wake_edge_connect(20 + j * 2 - 1)
Print #1, "side", "2", "1", duct_wake_edge_connect(j * 2 - 1)
Print #1, "side", "3", "1", duct_edge_connect(20 + j * 2 - 1)
Print #1, "sources", "0", "0", "0"

'fixed wake
face_count = face_count + 1
Print #1, "face", face_count, "4096", "3"
Print #1, "linear"
Print #1, "origin node", duct_wake_node_connect(j * 2 + 1)
Print #1, "side", "0", "1", duct_wake_edge_connect(j * 2 + 2)
Print #1, "side", "1", "1", duct_wake_edge_connect(20 + j * 2)
Print #1, "side", "2", "1", duct_wake_edge_connect(j * 2)
Print #1, "side", "3", "1", duct_wake_edge_connect(20 + j * 2 - 1)
Print #1, "sources", "0", "0", "0"

Next

End Sub


Public Sub duct_wake_nodes(node_count, duct_wake_node_connect)

Dim j As Integer

'duct wake nodes
'*******************************************
For j = 1 To 10
node_count = node_count + 1
duct_wake_node_connect(j * 2 - 1) = node_count
Print #1, "node", node_count, duct_wake(j, 2).x, duct_wake(j, 2).y, duct_wake(j, 2).z

node_count = node_count + 1
duct_wake_node_connect(j * 2) = node_count
Print #1, "node", node_count, duct_wake(j, 3).x, duct_wake(j, 3).y, duct_wake(j, 3).z

Next

End Sub
Public Sub duct_wake_edges(edge_count, duct_wake_node_connect,
duct_wake_edge_connect, duct_node_connect)
'duct wake edges
'**************************************************************
```

```
Dim i, j As Integer

For j = 1 To 10
'staight line to 1/3 of wake for adaption
edge_count = edge_count + 1
duct_wake_edge_connect(j * 2 - 1) = edge_count
If j = 1 Or j = 10 Then Print #1, "edge", edge_count, "-4", "2",
duct_freewake_panels_s, "0.3", "2.0"
If j <> 1 And j <> 10 Then Print #1, "edge", edge_count, "-1", "2",
duct_freewake_panels_s, "0.3", "2.0"
Print #1, "startnode", duct_node_connect(j * 4 - 3)
Print #1, "finishnode", duct_wake_node_connect(j * 2 - 1)


'staight line from 1/3  to end of wake
edge_count = edge_count + 1
duct_wake_edge_connect(j * 2) = edge_count
If j = 1 Or j = 10 Then Print #1, "edge", edge_count, "-4", "2",
duct_fixedwake_panels_s, "1.0", "0.1"
If j <> 1 And j <> 10 Then Print #1, "edge", edge_count, "-1", "2",
duct_fixedwake_panels_s, "1.0", "0.1"
Print #1, "startnode", duct_wake_node_connect(j * 2 - 1)
Print #1, "finishnode", duct_wake_node_connect(j * 2)

Next j

For j = 1 To 9

'first arc at 1/3 of wake for adaption
edge_count = edge_count + 1
duct_wake_edge_connect(20 + j * 2 - 1) = edge_count
Print #1, "edge", edge_count, "-1", "36", duct_panels_t, "1.0", "0.1"
Print #1, "startnode", duct_wake_node_connect(j * 2 - 1)
For i = 2 To 35
Print #1, (i - 1), duct_wake_arcs(j, 1, i).x, duct_wake_arcs(j, 1, i).y, duct_wake_arcs(j,
1, i).z
Next
Print #1, "finishnode", duct_wake_node_connect(j * 2 + 1)

'arc at end of wake
edge_count = edge_count + 1
duct_wake_edge_connect(20 + j * 2) = edge_count
Print #1, "edge", edge_count, "-5", "36", duct_panels_t, "1.0", "0.1"
Print #1, "startnode", duct_wake_node_connect(j * 2)
For i = 2 To 35
Print #1, (i - 1), duct_wake_arcs(j, 2, i).x, duct_wake_arcs(j, 2, i).y, duct_wake_arcs(j,
2, i).z
Next
Print #1, "finishnode", duct_wake_node_connect(j * 2 + 2)
```

```
Next j

End Sub
Public Sub duct_wakesheet(wake_length)
Dim i, j As Integer
ReDim duct_wake(10, 3)
ReDim duct_wake_arcs(10, 2, 36)

'set wake on the end of the duct. use upper section
For i = 1 To 10
duct_wake(i, 1).x = duct_upper(i, num_upper_points).x
duct_wake(i, 1).y = duct_upper(i, num_upper_points).y
duct_wake(i, 1).z = duct_upper(i, num_upper_points).z

'line to wake adapt
duct_wake(i, 2).x = duct_upper(i, num_upper_points).x
duct_wake(i, 2).y = duct_upper(i, num_upper_points).y + duct_freewake_length *
wake_length
duct_wake(i, 2).z = duct_upper(i, num_upper_points).z

'end of the straight lines and wake sheet
duct_wake(i, 3).x = duct_upper(i, num_upper_points).x
duct_wake(i, 3).y = duct_upper(i, num_upper_points).y + wake_length
duct_wake(i, 3).z = duct_upper(i, num_upper_points).z
Next

For j = 1 To 10
'duct wake arcs
For i = 1 To 36
'adaption arc
duct_wake_arcs(j, 1, i).x = duct_leading_arc(j, i).x
duct_wake_arcs(j, 1, i).y = duct_leading_arc(j, i).y + duct_freewake_length *
wake_length
duct_wake_arcs(j, 1, i).z = duct_leading_arc(j, i).z

'arc at the end of wake
duct_wake_arcs(j, 2, i).x = duct_leading_arc(j, i).x
duct_wake_arcs(j, 2, i).y = duct_leading_arc(j, i).y + wake_length
duct_wake_arcs(j, 2, i).z = duct_leading_arc(j, i).z
Next

Next
End Sub
Public Sub blade_faces(face_count, blade_node_connect, blade_ends_connect,
blade_section_connect, blade_edge_connect, hub_edge_connect, ring_edge_connect,
hub_node_connect)
Dim i As Integer
```

```
'blade faces
'*****************************************
For i = 1 To num_sections - 1
 'upper face
  face_count = face_count + 1
  Print #1, "face", face_count, "16", "1"
  Print #1, "linear"
  Print #1, "origin node", blade_node_connect((i - 1) * 2 + 2)
  Print #1, "side", "0", "1", blade_section_connect((i - 1) * 2 + 1)
  Print #1, "side", "1", "1", blade_edge_connect((i - 1) * 2 + 1)
  Print #1, "side", "2", "1", blade_section_connect((i - 1) * 2 + 3)
  Print #1, "side", "3", "1", blade_edge_connect((i - 1) * 2 + 2)
  Print #1, "sources", "0", "0", "0"

 'lower face
  face_count = face_count + 1
  Print #1, "face", face_count, "16", "1"
  Print #1, "linear"
  Print #1, "origin node", blade_node_connect((i - 1) * 2 + 1)
  Print #1, "side", "0", "1", blade_section_connect((i - 1) * 2 + 2)
  Print #1, "side", "1", "1", blade_edge_connect((i - 1) * 2 + 2)
  Print #1, "side", "2", "1", blade_section_connect((i - 1) * 2 + 4)
  Print #1, "side", "3", "1", blade_edge_connect((i - 1) * 2 + 1)
  Print #1, "sources", "0", "0", "0"

Next

End Sub
Public Sub pick_section_of_spline(curve() As section, sect_num, edge As String,
temp() As section, num_temp_points)

Dim i As Integer
Dim counter As Integer
Dim lower As Single
Dim upper As Single

If edge = "le" Then
 If final_section(sect_num, section_data(sect_num).le_id).x > final_section(sect_num
+ 1, section_data(sect_num + 1).le_id).x Then
  upper = final_section(sect_num, section_data(sect_num).le_id).x
  lower = final_section(sect_num + 1, section_data(sect_num + 1).le_id).x
 End If
 If final_section(sect_num, section_data(sect_num).le_id).x < final_section(sect_num
+ 1, section_data(sect_num + 1).le_id).x Then
  lower = final_section(sect_num, section_data(sect_num).le_id).x
  upper = final_section(sect_num + 1, section_data(sect_num + 1).le_id).x
 End If
End If

If edge = "te" Then
```

```
 If final_section(sect_num, section_data(sect_num).te_id).x > final_section(sect_num
+ 1, section_data(lower).te_id).x Then
  upper = final_section(sect_num, section_data(sect_num).te_id).x
  lower = final_section(sect_num + 1, section_data(sect_num + 1).te_id).x
 End If
 If final_section(sect_num, section_data(sect_num).te_id).x < final_section(sect_num
+ 1, section_data(lower).te_id).x Then
  lower = final_section(sect_num, section_data(sect_num).te_id).x
  upper = final_section(sect_num + 1, section_data(sect_num + 1).te_id).x
 End If
End If


counter = 0

For i = 1 To 101

If curve(i).x > lower And curve(i).x < upper Then
counter = counter + 1
temp(counter).x = curve(i).x
temp(counter).y = curve(i).y
temp(counter).z = curve(i).z
End If
Next

num_temp_points = counter

End Sub
Public Sub blade_edges(edge_count, blade_node_connect, blade_ends_connect,
blade_section_connect, blade_edge_connect, blade_panels_t, blade_panels_s,
hub_node_connect, ring_node_connect)
Dim num_upper_section_nodes, j, q, i As Integer

Dim lower, upper As Integer
Dim num_temp_points As Integer
Dim temp(101) As section

'*****************************************************************
******
'blade_section_connect definition
'*****************************************************************
******

For j = 1 To num_sections
  'lower side of section
  edge_count = edge_count + 1
  blade_section_connect((j - 1) * 2 + 1) = edge_count
  If j = 1 Then Print #1, "edge", edge_count, "-2", section_data(j).le_id,
blade_panels_s, 2 - blade_P, blade_Q
```

```
  If j <> 1 Then Print #1, "edge", edge_count, "-1", section_data(j).le_id,
blade_panels_s, 2 - blade_P, blade_Q
  Print #1, "startnode", blade_node_connect((j - 1) * 2 + 2)
 For i = 2 To section_data(j).le_id - 1
  Print #1, (i - 1), final_section(j, i).x, final_section(j, i).y, final_section(j, i).z
 Next
  Print #1, "finishnode", blade_node_connect((j - 1) * 2 + 1)

  'upper side of section
  edge_count = edge_count + 1
  blade_section_connect(((j - 1) * 2) + 2) = edge_count
  num_upper_section_nodes = 1 + section_data(j).num_points - section_data(j).le_id
  Print #1, "edge", edge_count, "-1", num_upper_section_nodes, blade_panels_s,
blade_P, blade_Q
  Print #1, "startnode", blade_node_connect((j - 1) * 2 + 1)
  q = section_data(j).le_id
 For i = 2 To num_upper_section_nodes - 1
    q = q + 1
    Print #1, (i - 1), final_section(j, q).x, final_section(j, q).y, final_section(j, q).z
 Next
  Print #1, "finishnode", blade_node_connect((j - 1) * 2 + 2)
Next

'*****************************************************************
******
'blade_edge_connect definition
'*****************************************************************
******
Dim g As Integer

j = 0
For i = 1 To num_sections - 1

'leading edge
Call pick_section_of_spline(leading_spline(), i, "le", temp(), num_temp_points)
j = j + 1
edge_count = edge_count + 1
blade_edge_connect(j) = edge_count
If i = (num_sections - 1) Then Print #1, "edge", edge_count, "-1", (num_temp_points
+ 2), Int(blade_panels_t * blade_tip_cluster), "1.0", "0.1"
If i <> (num_sections - 1) Then Print #1, "edge", edge_count, "-1", (num_temp_points
+ 2), blade_panels_t, "1.0", "0.1"
Print #1, "startnode", blade_node_connect((i - 1) * 2 + 1)
For g = 1 To num_temp_points
   Print #1, (g), temp(g).x, temp(g).y, temp(g).z
Next
Print #1, "finishnode", blade_node_connect((i - 1) * 2 + 3)

'trailing edge
Call pick_section_of_spline(trailing_spline(), i, "te", temp(), num_temp_points)
```

```
j = j + 1
edge_count = edge_count + 1
blade_edge_connect(j) = edge_count
If i = (num_sections - 1) Then Print #1, "edge", edge_count, "-3", (num_temp_points
+ 2), Int(blade_panels_t * blade_tip_cluster), "1.0", "0.1"
If i <> (num_sections - 1) Then Print #1, "edge", edge_count, "-3", (num_temp_points
+ 2), blade_panels_t, "1.0", "0.1"
Print #1, "startnode", blade_node_connect((i - 1) * 2 + 2)
For g = 1 To num_temp_points
  Print #1, (g), temp(g).x, temp(g).y, temp(g).z
Next

Print #1, "finishnode", blade_node_connect((i - 1) * 2 + 4)
Next

End Sub
Public Sub blade_nodes(node_count, blade_node_connect)
Dim i As Integer

'blade nodes
'*****************************************************************
*****
For i = 1 To num_sections

'section edges
node_count = node_count + 1
blade_node_connect(((i - 1) * 2) + 1) = node_count
Print #1, "node", node_count, final_section(i, section_data(i).le_id).x, final_section(i,
section_data(i).le_id).y, final_section(i, section_data(i).le_id).z
node_count = node_count + 1
blade_node_connect(((i - 1) * 2) + 2) = node_count
Print #1, "node", node_count, final_section(i, section_data(i).te_id).x, final_section(i,
section_data(i).te_id).y, final_section(i, section_data(i).te_id).z

Next
End Sub
Public Sub missing_hub_nodes(node_count, blade_node_connect)
'nodes at the root needed for the hub when the blade is not exported
'*****************************************************************
*****
'section edges
node_count = node_count + 1
blade_node_connect(1) = node_count
Print #1, "node", node_count, final_section(1, section_data(1).le_id).x,
final_section(1, section_data(1).le_id).y, final_section(1, section_data(1).le_id).z
node_count = node_count + 1
blade_node_connect(2) = node_count
Print #1, "node", node_count, final_section(1, section_data(1).te_id).x,
final_section(1, section_data(1).te_id).y, final_section(1, section_data(1).te_id).z
```

```
End Sub
Public Sub hub_faces(face_count, hub_node_connect, hub_edge_connect,
blade_section_connect, blade_edge_connect)
'*******************************************************************
****



Dim i As Integer

'hub leading faces
For i = 1 To no_of_hub_strips
face_count = face_count + 1
Print #1, "face", face_count, "16", "1"
Print #1, "linear"
Print #1, "origin node", hub_node_connect(i + 1)
Print #1, "side", "0", "1", hub_edge_connect(10 + 1 + i)
Print #1, "side", "1", "1", hub_edge_connect(20 + i)
Print #1, "side", "2", "1", hub_edge_connect(10 + i)
Print #1, "side", "3", "1", hub_edge_connect(i)
Print #1, "sources", "0", "0", "0"
Next

'hub faces between blades
For i = 1 To no_of_hub_strips
face_count = face_count + 1
Print #1, "face", face_count, "16", "1"
Print #1, "linear"
Print #1, "origin node", hub_node_connect(i + 1 + 10)
Print #1, "side", "0", "1", hub_edge_connect(30 + 1 + i)
Print #1, "side", "1", "1", hub_edge_connect(40 + i)
If i = 1 Then Print #1, "side", "2", "1", blade_section_connect(2)
If i <> 1 Then Print #1, "side", "2", "1", hub_edge_connect(30 + i)
Print #1, "side", "3", "1", hub_edge_connect(20 + i)
Print #1, "sources", "0", "0", "0"
Next
'hub trailing faces
For i = 1 To no_of_hub_strips
face_count = face_count + 1
Print #1, "face", face_count, "16", "1"
Print #1, "linear"
Print #1, "origin node", hub_node_connect(i + 21)
Print #1, "side", "0", "1", hub_edge_connect(50 + 1 + i)
Print #1, "side", "1", "1", hub_edge_connect(60 + i)
Print #1, "side", "2", "1", hub_edge_connect(50 + i)
Print #1, "side", "3", "1", hub_edge_connect(40 + i)
Print #1, "sources", "0", "0", "0"
Next
```

```
End Sub

Public Sub ring_faces(face_count, ring_node_connect, ring_edge_connect,
blade_section_connect)


'*****************************************************************
****
'ring  face

face_count = face_count + 1
Print #1, "face", face_count, "16", "1"
Print #1, "linear"
Print #1, "origin node", ring_node_connect(1)
Print #1, "side", "0", "1", ring_edge_connect(3)
Print #1, "side", "1", "1", ring_edge_connect(1)
Print #1, "side", "2", "1", blade_section_connect(num_sections * 2)
Print #1, "side", "3", "1", ring_edge_connect(2)
Print #1, "sources", "0", "0", "0"

End Sub

Public Sub caps(hub_length)
Dim no_of_points As Integer
Dim i As Integer
Dim radius, offset As Single
Dim revolution_angle As Single

Dim cap_internal_y2() As section

'no of points for back section
'created automatically
no_of_points = 36



Dim scale_factor As Single

'radius of hub
radius = Sqr(final_section(1, section_data(1).le_id).x ^ 2 + final_section(1,
section_data(1).le_id).z ^ 2)


'positon of cap according to hub length
offset = hub_length / 2

'should be the same with the number of stators
no_of_blades = 4
revolution_angle = 2 * pi / (duct_images * 2)
```

```
If cap_auto = 1 Then
cap_section_points = 36
ReDim cap_left_temp(cap_section_points)
'automatically create an arc to make a spherical cap
Call arc(cap_left_temp(), 0, pi / 2, radius, 0, offset, 0, cap_section_points, 2)
Else
'match cap radius with hub radius
scale_factor = radius / cap_left_temp(cap_section_points).x

'otherwise already read
For i = 1 To cap_section_points
cap_left_temp(i).x = cap_left_temp(i).x * scale_factor
cap_left_temp(i).y = cap_left_temp(i).y * scale_factor + offset
cap_left_temp(i).z = cap_left_temp(i).z * scale_factor
Next
End If



'calculate total length of arc on hub
Dim total_length As Single
total_length = 0
For i = 1 To cap_section_points - 1
total_length = total_length + Sqr((cap_left_temp(i).x - cap_left_temp(i + 1).x) ^ 2 +
(cap_left_temp(i).y - cap_left_temp(i + 1).y) ^ 2 + (cap_left_temp(i).z -
cap_left_temp(i + 1).z) ^ 2)
Next

'fit a spline throught cap_left
'*******************************************************************
***
'calculate no of spline points for 20 points on internal
Dim distance As Single
Dim points_per_distance As Single
Dim num_of_spline_points As Integer

distance = (side_fraction - internal_fraction) * total_length
points_per_distance = cap_section_points / distance
num_of_spline_points = Abs(Int(total_length * points_per_distance))

'call spline routine

ReDim cap_left(num_of_spline_points + 1)
ReDim cap_internal_y2(num_of_spline_points + 1)
ReDim cap_right(num_of_spline_points + 1)
ReDim cap_back(no_of_points)
ReDim cap_internal_y(num_of_spline_points + 1)
ReDim cap_internal_x(num_of_spline_points + 1)
ReDim cap_internal_z(num_of_spline_points + 1)
```

```
Call s_spline(cap_left_temp(), 1, 1, cap_left(), cap_section_points,
num_of_spline_points)
cap_section_points = num_of_spline_points + 1


'arc at the back of cap
Call arc(cap_back(), pi / 2, pi / 2 + revolution_angle, radius, 0, offset, 0,
no_of_points, 1)

'other arc of cap depending on no_of_stators
For i = 1 To cap_section_points
cap_right(i).x = cap_left(i).x * Cos(revolution_angle) + cap_left(i).z *
Sin(revolution_angle)
cap_right(i).y = cap_left(i).y
cap_right(i).z = -cap_left(i).x * Sin(revolution_angle) + cap_left(i).z *
Cos(revolution_angle)
Next

'calculate new total length of arc on hub
'more accurate since spline
Dim length() As Single
ReDim length(cap_section_points)
total_length = 0
length(0) = 0
length(1) = 0
For i = 2 To cap_section_points
length(i) = length(i - 1) + Sqr((cap_left(i).x - cap_left(i - 1).x) ^ 2 + (cap_left(i).y -
cap_left(i - 1).y) ^ 2 + (cap_left(i).z - cap_left(i - 1).z) ^ 2)
Next
total_length = length(cap_section_points)


Dim break As Integer
'find id for point half-way along the line
break = 0
For i = 1 To cap_section_points

If length(i) > (total_length * (1 - side_fraction)) And break = 0 Then
cap_sidenode_id = i
break = 1
End If

Next




Dim counter As Integer
Dim length_to_point As Single
Dim x1, y1, z1 As Single
Dim x2, y2, z2 As Single
Dim d_length As Single
Dim length_fraction As Single

'Internal arc along hub axis
For i = 1 To cap_section_points

If length(i) > (total_length * (1 - internal_fraction)) Then
'interpolate first point
 If counter = 0 Then

   counter = counter + 1

   x1 = cap_left(i - 1).x * Cos(revolution_angle / 2) + cap_left(i - 1).z *
Sin(revolution_angle / 2)
   y1 = cap_left(i - 1).y
   z1 = -cap_left(i - 1).x * Sin(revolution_angle / 2) + cap_left(i - 1).z *
Cos(revolution_angle / 2)

   x2 = cap_left(i).x * Cos(revolution_angle / 2) + cap_left(i).z *
Sin(revolution_angle / 2)
   y2 = cap_left(i).y
   z2 = -cap_left(i).x * Sin(revolution_angle / 2) + cap_left(i).z *
Cos(revolution_angle / 2)

   length_fraction = 1 - (length(i) - (total_length * (1 - internal_fraction))) / (length(i)
- length(i - 1))

   cap_internal_x(counter).x = x1 + (x2 - x1) * length_fraction
   cap_internal_x(counter).y = y1 + (y2 - y1) * length_fraction
   cap_internal_x(counter).z = z1 + (z2 - z1) * length_fraction
 End If

 counter = counter + 1
 cap_internal_x(counter).x = cap_left(i).x * Cos(revolution_angle / 2) + cap_left(i).z
* Sin(revolution_angle / 2)
 cap_internal_x(counter).y = cap_left(i).y
 cap_internal_x(counter).z = -cap_left(i).x * Sin(revolution_angle / 2) + cap_left(i).z
* Cos(revolution_angle / 2)

End If
Next

num_internal_x_points = counter
counter = 0

'calculate length of internal x
Dim internal_length As Single
internal_length = 0
For i = 1 To num_internal_x_points - 1
```

```
internal_length = internal_length + Sqr((cap_internal_x(i).x - cap_internal_x(i + 1).x)
^ 2 + (cap_internal_x(i).y - cap_internal_x(i + 1).y) ^ 2 + (cap_internal_x(i).z -
cap_internal_x(i + 1).z) ^ 2)
Next


'calculate the other two internal sections on cap
Dim angle As Single


For i = 1 To cap_section_points

If length(i) > length(cap_sidenode_id) And (length(i) < (total_length -
internal_length)) Then
 angle = (revolution_angle / 2) * ((length(i) - length(cap_sidenode_id)) / (total_length
- length(cap_sidenode_id) - internal_length))
 counter = counter + 1
 cap_internal_y(counter).x = cap_left(i).x * Cos(angle) + cap_left(i).z * Sin(angle)
 cap_internal_y(counter).y = cap_left(i).y
 cap_internal_y(counter).z = -cap_left(i).x * Sin(angle) + cap_left(i).z * Cos(angle)

 cap_internal_z(counter).x = cap_left(i).x * Cos(revolution_angle - angle) +
cap_left(i).z * Sin(revolution_angle - angle)
 cap_internal_z(counter).y = cap_left(i).y
 cap_internal_z(counter).z = -cap_left(i).x * Sin(revolution_angle - angle) +
cap_left(i).z * Cos(revolution_angle - angle)
End If

Next
num_internal_points = counter

'rotate cap to match hub
Dim hub_angle As Single
Dim angle_r As Single
Dim cap_angle As Single

hub_angle = get_angle(hub_trailing_edge(1).x, hub_trailing_edge(1).z)
cap_angle = get_angle(cap_back(1).x, cap_back(1).z)

angle_r = hub_angle - cap_angle
Call rotate(cap_left(), angle_r, cap_section_points)
Call rotate(cap_back(), angle_r, no_of_points)
Call rotate(cap_right(), angle_r, cap_section_points)
Call rotate(cap_internal_x(), angle_r, num_internal_x_points)
Call rotate(cap_internal_y(), angle_r, num_internal_points)
Call rotate(cap_internal_z(), angle_r, num_internal_points)
End Sub
Public Sub rotate(v() As section, angle, num_points)
Dim i As Integer
Dim temp() As section
```

```
ReDim temp(num_points)

For i = 1 To num_points
temp(i).x = v(i).x
temp(i).z = v(i).z
Next

For i = 1 To num_points
 v(i).x = temp(i).x * Cos(angle) + temp(i).z * Sin(angle)
 v(i).z = -temp(i).x * Sin(angle) + temp(i).z * Cos(angle)
Next


End Sub
Public Sub rotate_point(v As section, angle, num_points)
Dim i As Integer
Dim temp As section

temp.x = v.x
temp.z = v.z


For i = 1 To num_points
 v.x = temp.x * Cos(angle) + temp.z * Sin(angle)
 v.z = -temp.x * Sin(angle) + temp.z * Cos(angle)
Next


End Sub

Private Function get_angle(x As Single, z As Single) As Single
Dim f As Single
Dim angle As Single

f = x / z
Select Case f

Case Is > 0
If x > 0 Then angle = Atn(Abs(f))
If x < 0 Then angle = pi + Atn(Abs(f))
Case Is < 0
If x > 0 Then angle = pi - Atn(Abs(f))
If x < 0 Then angle = 2 * pi - Atn(Abs(f))
Case Is = 0
If z > 0 Then angle = 0
If z < 0 Then angle = pi
End Select
get_angle = angle
End Function
```

```
Public Sub duct()

Dim radius, x_offset, duct_rev_angle As Single
Dim num_duct_arc_points As Integer
Dim i, j As Integer
Dim start_angle, end_angle, offset As Single
Dim duct_rev_angle_step As Single

ReDim duct_trailing_lower(10, section_data(0).le_id)
ReDim duct_leading_lower(10, section_data(0).le_id)
num_upper_points = section_data(0).num_points - section_data(0).le_id + 1
ReDim duct_upper(10, num_upper_points)
ReDim duct_lower(10, section_data(0).le_id)

'scale section
For i = 1 To section_data(0).num_points
section(0, i).x = section(0, i).x * duct_length * D
section(0, i).y = section(0, i).y * duct_length * (duct_thickness / 0.1) * D
Next


num_leading_lower_points = 0
num_trailing_lower_points = 0
'track down the lower side of the duct section and remove the ring width
For i = 1 To section_data(0).le_id
'lower sections for duct without ring
duct_lower(1, i).x = -section(0, i).y
duct_lower(1, i).y = section(0, i).x
duct_lower(1, i).z = section(0, i).z

'lower section for duct with ring
'aft part of lower ducts section minus the ring width
If section(0, i).x <= (final_section(num_sections, section_data(num_sections).le_id).y
- ring_width * D / 100) Then
  'interpolate for the point on the ring_width
  If section(0, i - 1).x > (final_section(num_sections,
section_data(num_sections).le_id).y - ring_width * D / 100) Then
    num_trailing_lower_points = num_trailing_lower_points + 1
    duct_trailing_lower(1, num_trailing_lower_points).y = final_section(num_sections,
section_data(num_sections).le_id).y - ring_width * D / 100
    duct_trailing_lower(1, num_trailing_lower_points).x = -(((section(0, i - 1).y -
section(0, i).y) / (section(0, i - 1).x - section(0, i).x)) * ((final_section(num_sections,
section_data(num_sections).le_id).y) - ring_width * D / 100 - section(0, i).x) +
section(0, i).y)
    duct_trailing_lower(1, num_trailing_lower_points).z = 0
  End If
 num_trailing_lower_points = num_trailing_lower_points + 1
 duct_trailing_lower(1, num_trailing_lower_points).y = section(0, i).x
 duct_trailing_lower(1, num_trailing_lower_points).x = -section(0, i).y
 duct_trailing_lower(1, num_trailing_lower_points).z = 0
End If
'forward part of lower ducts section minus the ring width
If section(0, i).x >= (final_section(num_sections, section_data(num_sections).te_id).y
+ ring_width * D / 100) Then
 num_leading_lower_points = num_leading_lower_points + 1
 duct_leading_lower(1, num_leading_lower_points).y = section(0, i).x
 duct_leading_lower(1, num_leading_lower_points).x = -section(0, i).y
 duct_leading_lower(1, num_leading_lower_points).z = 0

 'interpolate for the point on the ring_width
 If section(0, i + 1).x < (final_section(num_sections,
section_data(num_sections).te_id).y + ring_width * D / 100) Then
   num_leading_lower_points = num_leading_lower_points + 1
   duct_leading_lower(1, num_leading_lower_points).y = final_section(num_sections,
section_data(num_sections).te_id).y + ring_width * D / 100
   duct_leading_lower(1, num_leading_lower_points).x = -(((section(0, i + 1).y -
section(0, i).y) / (section(0, i + 1).x - section(0, i).x)) * (final_section(num_sections,
section_data(num_sections).te_id).y + ring_width * D / 100 - section(0, i).x) +
section(0, i).y)
   duct_leading_lower(1, num_leading_lower_points).z = 0
 End If
End If
Next

'upper part of duct section

num_upper_points = 0
For i = section_data(0).le_id To section_data(0).num_points
num_upper_points = num_upper_points + 1
duct_upper(1, num_upper_points).y = section(0, i).x
duct_upper(1, num_upper_points).x = -section(0, i).y
duct_upper(1, num_upper_points).z = 0
Next


'move section to correct radius
radius = D / 2 'Sqr(final_section(num_sections, section_data(1).le_id).x ^ 2 +
final_section(num_sections, section_data(1).le_id).z ^ 2)
x_offset = radius '- duct_trailing_lower(1, 1).x

For i = 1 To num_leading_lower_points
duct_leading_lower(1, i).x = duct_leading_lower(1, i).x + x_offset
Next

For i = 1 To num_trailing_lower_points
 duct_trailing_lower(1, i).x = duct_trailing_lower(1, i).x + x_offset
Next

For i = 1 To num_upper_points
```

```
 duct_upper(1, i).x = duct_upper(1, i).x + x_offset
Next

'lower sections for duct without ring
For i = 1 To section_data(0).le_id
duct_lower(1, i).x = duct_lower(1, i).x + x_offset
Next



'''other duct sections that are rotated
'duct_images = 4
'9 is the number of strips for duct ie 10 sections
'cannot use one because the shape is not circular
duct_rev_angle_step = -2 * pi / duct_images / 9

For j = 1 To 9
duct_rev_angle = duct_rev_angle + duct_rev_angle_step
For i = 1 To num_leading_lower_points
duct_leading_lower(j + 1, i).y = duct_leading_lower(1, i).y
duct_leading_lower(j + 1, i).x = Cos(duct_rev_angle) * duct_leading_lower(1, i).x +
Sin(duct_rev_angle) * duct_leading_lower(1, i).z
duct_leading_lower(j + 1, i).z = -Sin(duct_rev_angle) * duct_leading_lower(1, i).x +
Cos(duct_rev_angle) * duct_leading_lower(1, i).z
Next

For i = 1 To num_trailing_lower_points
duct_trailing_lower(j + 1, i).y = duct_trailing_lower(1, i).y
duct_trailing_lower(j + 1, i).x = Cos(duct_rev_angle) * duct_trailing_lower(1, i).x +
Sin(duct_rev_angle) * duct_trailing_lower(1, i).z
duct_trailing_lower(j + 1, i).z = -Sin(duct_rev_angle) * duct_trailing_lower(1, i).x +
Cos(duct_rev_angle) * duct_trailing_lower(1, i).z
Next

For i = 1 To num_upper_points
duct_upper(j + 1, i).y = duct_upper(1, i).y
duct_upper(j + 1, i).x = Cos(duct_rev_angle) * duct_upper(1, i).x +
Sin(duct_rev_angle) * duct_upper(1, i).z
duct_upper(j + 1, i).z = -Sin(duct_rev_angle) * duct_upper(1, i).x +
Cos(duct_rev_angle) * duct_upper(1, i).z
Next

'seconf lower section for duct without ring
For i = 1 To section_data(0).le_id
duct_lower(j + 1, i).y = duct_lower(1, i).y
duct_lower(j + 1, i).x = Cos(duct_rev_angle) * duct_lower(1, i).x +
Sin(duct_rev_angle) * duct_lower(1, i).z
duct_lower(j + 1, i).z = -Sin(duct_rev_angle) * duct_lower(1, i).x +
Cos(duct_rev_angle) * duct_lower(1, i).z
Next
```

```
Next j

'draw arcs of duct
num_duct_arc_points = 36
ReDim duct_leading_arc(10, num_duct_arc_points)
ReDim duct_trailing_arc(10, num_duct_arc_points)
ReDim duct_leading_lower_arc(10, num_duct_arc_points)
ReDim duct_trailing_lower_arc(10, num_duct_arc_points)

start_angle = pi / 2 - duct_rev_angle_step
For j = 1 To 9

start_angle = start_angle + duct_rev_angle_step
end_angle = start_angle + duct_rev_angle_step
'trailing edge arc
radius = Sqr(duct_upper(1, 1).x ^ 2 + duct_upper(1, 1).z ^ 2)
offset = duct_upper(1, 1).y
Call arcs(duct_trailing_arc(), start_angle, end_angle, radius, 0, offset, 0,
num_duct_arc_points, 1, j)

'leading edge arc
radius = Sqr(duct_leading_lower(1, 1).x ^ 2 + duct_leading_lower(1, 1).z ^ 2)
offset = duct_leading_lower(1, 1).y
Call arcs(duct_leading_arc(), start_angle, end_angle, radius, 0, offset, 0,
num_duct_arc_points, 1, j)

'trailing lower edge arc
radius = Sqr(duct_trailing_lower(1, 1).x ^ 2 + duct_trailing_lower(1, 1).z ^ 2)
offset = duct_trailing_lower(1, 1).y
Call arcs(duct_trailing_lower_arc(), start_angle, end_angle, radius, 0, offset, 0,
num_duct_arc_points, 1, j)

'leading lower edge arc
radius = Sqr(duct_leading_lower(1, num_leading_lower_points).x ^ 2 +
duct_leading_lower(1, num_leading_lower_points).z ^ 2)
offset = duct_leading_lower(1, num_leading_lower_points).y
Call arcs(duct_leading_lower_arc(), start_angle, end_angle, radius, 0, offset, 0,
num_duct_arc_points, 1, j)
Next j
End Sub


Public Sub ring_leading_section(ring_width)
Dim num_edge_points, i, j As Integer
Dim psi, dpsi, r, P As Single
Dim angle, angle_step As Single


num_edge_points = 100
```

```
ReDim ring_leading_end(10, num_edge_points)

'set point to trailing edge of first section
ring_leading_end(1, 1).x = final_section(num_sections,
section_data(num_sections).le_id).x
ring_leading_end(1, 1).y = final_section(num_sections,
section_data(num_sections).le_id).y
ring_leading_end(1, 1).z = final_section(num_sections,
section_data(num_sections).le_id).z

r = Sqr(ring_leading_end(1, 1).z ^ 2 + ring_leading_end(1, 1).x ^ 2)


P = propdata(num_sections).pitch * D
If P = 0 Then P = 0.1
'calculate the start angle and the step angle

'calculate start psi so it matches with leading edge of section
psi = (final_section(num_sections, section_data(num_sections).le_id).y / P) * 2 * pi
'step dpsi for the set number of steps
dpsi = ((((-ring_width / 2) - ring_leading_end(1, 1).y) / P) * 2 * pi) /
(num_edge_points - 1)
psi = psi ' - dpsi
'helical edge
'*********************
For i = 2 To num_edge_points
psi = psi + dpsi
ring_leading_end(1, i).x = r * Cos(psi)
ring_leading_end(1, i).y = (P * psi) / (2 * pi)
ring_leading_end(1, i).z = -r * Sin(psi)
Next

End Sub

Public Sub hub_trailing_section(hub_length)
Dim psi, dpsi, r As Single
Dim P As Single
Dim num_edge_points, i, j As Integer
Dim angle

num_edge_points = 100
ReDim hub_trailing_end(10, num_edge_points)

'set point to trailing edge of section
hub_trailing_end(1, 1).x = final_section(1, section_data(1).te_id).x
hub_trailing_end(1, 1).y = final_section(1, section_data(1).te_id).y
hub_trailing_end(1, 1).z = final_section(1, section_data(1).te_id).z

r = Sqr(hub_trailing_end(1, 1).z ^ 2 + hub_trailing_end(1, 1).x ^ 2)
```

```
'set the pitch of the hub to the wake and not the blade so
'there are no problems with wake influence on hub
P = find_trailing_edge_pitch(1)
If P = 0 Then P = 0.01

'calculate dpsi such that the hub length is correct
dpsi = ((((hub_length / 2) - hub_trailing_end(1, 1).y) / (P * D)) * 2 * pi) /
(num_edge_points - 1)


'helical edge
'*********************
For i = 2 To num_edge_points
'use te as starting point to ensure perfect match
hub_trailing_end(1, i).x = hub_trailing_end(1, i - 1).x * Cos(-dpsi) -
hub_trailing_end(1, i - 1).z * Sin(-dpsi)
hub_trailing_end(1, i).z = hub_trailing_end(1, i - 1).x * Sin(-dpsi) +
hub_trailing_end(1, i - 1).z * Cos(-dpsi)
hub_trailing_end(1, i).y = hub_trailing_end(1, i - 1).y + (P * D * dpsi) / (2 * pi)

Next


End Sub
Public Sub ring_trailing_section(ring_width)
Dim psi, dpsi, r As Single
Dim P As Single
Dim num_edge_points, i, j As Integer
Dim angle

num_edge_points = 100
ReDim ring_trailing_end(10, num_edge_points)

'set point to trailing edge of section
ring_trailing_end(1, 1).x = final_section(num_sections,
section_data(num_sections).te_id).x
ring_trailing_end(1, 1).y = final_section(num_sections,
section_data(num_sections).te_id).y
ring_trailing_end(1, 1).z = final_section(num_sections,
section_data(num_sections).te_id).z

r = Sqr(ring_trailing_end(1, 1).z ^ 2 + ring_trailing_end(1, 1).x ^ 2)

P = propdata(num_sections).pitch
If P = 0 Then P = 0.1

'calculate the starting psi so it matches the trailing end
psi = (final_section(num_sections, section_data(num_sections).te_id).y / P) * 2 * pi
```

```
dpsi = ((((ring_width / 2) - ring_trailing_end(1, 1).y) / P) * 2 * pi) / (num_edge_points
- 1)
psi = psi ' - dpsi

'helical edge
'*********************
For i = 2 To num_edge_points
psi = psi + dpsi
ring_trailing_end(1, i).x = r * Cos(psi)
ring_trailing_end(1, i).y = (P * psi) / (2 * pi)
ring_trailing_end(1, i).z = -r * Sin(psi)
Next


End Sub

Public Sub ring_arc()
Dim no_of_points As Integer
Dim x1, y1, x2, y2 As Single
Dim radius As Single
Dim angle_of_revolution As Double
Dim angle, step_angle As Single
Dim start_angle, end_angle As Single
Dim f As Single

no_of_points = 121

ReDim ring_leading_edge(no_of_points)
ReDim ring_trailing_edge(no_of_points)
'Call calculate_trailing_end_of_ring(no_of_points)
'Call calculate_leading_end_of_ring(no_of_points)
'Call calculate_second_end_of_ring
radius = Sqr(final_section(num_sections, section_data(num_sections).le_id).x ^ 2 +
final_section(num_sections, section_data(num_sections).le_id).z ^ 2)

'set beginning of arc for ring
f = (final_section(num_sections, section_data(num_sections).te_id).z / radius)
'position of end on ring

Select Case f
Case Is = -1
If f > 0 Then start_angle = pi
If f < 0 Then start_angle = 0
Case Is = 1
If f > 0 Then start_angle = 0
If f < 0 Then start_angle = pi
Case Is <> 1, -1
'If f > 0 Then
start_angle = pi / 2 - asin(f)
'If f < 0 Then start_angle = pi / 2 + asin(f)
```

```
End Select
end_angle = start_angle + 2 * pi / no_of_blades
'draws arc
Call arc(ring_trailing_edge(), start_angle, end_angle, radius, 0,
final_section(num_sections, section_data(num_sections).te_id).y, 0, no_of_points, 1)

'other arc of ring

f = (final_section(num_sections, section_data(num_sections).le_id).z / radius)
'position of end on ring
Select Case f
Case Is = -1
If f > 0 Then start_angle = 0
If f < 0 Then start_angle = pi
Case Is = 1
If f > 0 Then start_angle = 0
If f < 0 Then start_angle = pi
Case Is <> 1, -1
If f > 0 Then start_angle = pi / 2 - asin(f)
If f < 0 Then start_angle = 3 * pi / 2 + asin(f)
End Select
end_angle = start_angle + 2 * pi / no_of_blades
'draws arc
Call arc(ring_leading_edge(), start_angle, end_angle, radius, 0,
final_section(num_sections, section_data(num_sections).le_id).y, 0, no_of_points, 1)
End Sub
Public Sub arcs(cyclos() As section, start_angle, end_angle, radius, x_offset, y_offset,
z_offset, no_of_points, Axis, section_number As Integer)
'draws arcs given a start and finish angle
'same as arc procedure but has two dimensional array support
'axis decides avout which axis
'0 is x, 1 is y, 2 is z
'x_offset etc is the position of the centre

Dim i As Integer
Dim angle_of_revolution, step_angle, angle As Single

step_angle = (end_angle - start_angle) / (no_of_points - 1)
angle = start_angle - step_angle
If Axis = 0 Then
For i = 1 To no_of_points
 angle = angle + step_angle
 cyclos(section_number, i).x = x_offset
 cyclos(section_number, i).y = Sin(angle) * radius + y_offset
 cyclos(section_number, i).z = Cos(angle) * radius + z_offset
Next
End If

If Axis = 1 Then
For i = 1 To no_of_points
```

```
 angle = angle + step_angle
 cyclos(section_number, i).x = Sin(angle) * radius + x_offset
 cyclos(section_number, i).y = y_offset
 cyclos(section_number, i).z = Cos(angle) * radius + z_offset
Next
End If

If Axis = 2 Then
For i = 1 To no_of_points
 angle = angle + step_angle
 cyclos(section_number, i).x = Sin(angle) * radius + x_offset
 cyclos(section_number, i).y = Cos(angle) * radius + y_offset
 cyclos(section_number, i).z = z_offset

Next
End If


End Sub

Public Sub arc(cyclos() As section, start_angle, end_angle, radius, x_offset, y_offset,
z_offset, no_of_points, Axis)
'draws arcs given a start and finish angle
'axis decides about which axis
'0 is x, 1 is y, 2 is z
'x_offset etc is the position of the centre

Dim i As Integer
Dim angle_of_revolution, step_angle, angle As Single

step_angle = (end_angle - start_angle) / (no_of_points - 1)
angle = start_angle - step_angle
If Axis = 0 Then
For i = 1 To no_of_points
 angle = angle + step_angle
 cyclos(i).x = x_offset
 cyclos(i).y = Sin(angle) * radius + y_offset
 cyclos(i).z = Cos(angle) * radius + z_offset
Next
End If

If Axis = 1 Then
For i = 1 To no_of_points
 angle = angle + step_angle
 cyclos(i).x = Sin(angle) * radius + x_offset
 cyclos(i).y = y_offset
 cyclos(i).z = Cos(angle) * radius + z_offset
Next
End If
```

```
If Axis = 2 Then
For i = 1 To no_of_points
 angle = angle + step_angle
 cyclos(i).x = Sin(angle) * radius + x_offset
 cyclos(i).y = Cos(angle) * radius + y_offset
 cyclos(i).z = z_offset

Next
End If


End Sub
Private Sub calculate_trailing_end_of_ring(num_edge_points)
Dim psi, phi, dpsi, r As Single
Dim P As Single
Dim i As Integer

ReDim ring_trailing_end(num_edge_points)

'set point to trailing edge of section
ring_trailing_end(1).x = final_section(num_sections,
section_data(num_sections).te_id).x
ring_trailing_end(1).y = final_section(num_sections,
section_data(num_sections).te_id).y
ring_trailing_end(1).z = final_section(num_sections,
section_data(num_sections).te_id).z

r = Sqr(ring_trailing_end(1).z ^ 2 + ring_trailing_end(1).x ^ 2)

'Calculate phi
P = propdata(num_sections).pitch
If P = 0 Then P = 0.1
phi = Atn(2 * pi * r / P)
psi = section(num_sections, section_data(num_sections).te_id).x / Sqr(r ^ 2 + (P / (2 *
pi)) ^ 2)
dpsi = ((((ring_width / 2) - ring_trailing_end(1).y) / P) * 2 * pi) / (num_edge_points -
1)
psi = psi ' - dpsi
'helical edge
'*********************
For i = 2 To num_edge_points
psi = psi + dpsi
ring_trailing_end(i).x = r * Cos(psi)
ring_trailing_end(i).y = (P * psi) / (2 * pi)
ring_trailing_end(i).z = -r * Sin(psi)
Next
End Sub
Public Sub second_hub_end()
'routine that claculates the second end of the hub
'it takes the two helixs from and rotates them by 390/no_of_blades
```

```
'and half the section of the root blade
Dim i, j As Integer
Dim angle As Single
num_hub_section_points = section_data(1).le_id + 1
ReDim hub_section_edge(10, num_hub_section_points)

angle = 2 * pi / no_of_blades

For i = 1 To section_data(1).le_id
hub_section_edge(10, i).x = Cos(angle) * final_section(1, i).x + Sin(angle) *
final_section(1, i).z
hub_section_edge(10, i).y = final_section(1, i).y
hub_section_edge(10, i).z = -Sin(angle) * final_section(1, i).x + Cos(angle) *
final_section(1, i).z
Next
End Sub
Public Sub second_ring_end()
'routine that calculates the section on the other side of the ring

Dim i, j As Integer
Dim angle As Single
num_ring_section_points = Abs(section_data(num_sections).num_points -
section_data(num_sections).le_id) + 1
ReDim ring_section_edge(num_ring_section_points)

angle = 2 * pi / no_of_blades

For i = 1 To section_data(num_sections).le_id
ring_section_edge(i).x = Cos(angle) * final_section(num_sections, i).x + Sin(angle) *
final_section(num_sections, i).z
ring_section_edge(i).y = final_section(num_sections, i).y
ring_section_edge(i).z = -Sin(angle) * final_section(num_sections, i).x + Cos(angle) *
final_section(num_sections, i).z
Next
End Sub

Public Sub wakesheet(wake_length)
Dim i, j As Integer
Dim variable_P, average_P, P As Single
Dim psi, dpsi, r As Single
Dim contraction_factor As Single
Dim start_psi, y_offset As Single
Dim wake_r As Single
Dim xsi As Single
Dim fun As Single
Dim slip_ratio As Single
Dim advance_ratio As Single
Dim n As Single
Dim pw As Single
```

```
no_of_wake_points = 300
ReDim wake(num_sections, no_of_wake_points)

transition_length = transition_length * D
'find average_pitch
For j = 1 To num_sections
average_P = average_P + propdata(j).pitch
Next
average_P = average_P / num_sections


'advance speed

advance_ratio = va / (rps * D)
slip_ratio = 1 - advance_ratio / average_P

'final wake contraction
wake_r = (1 - wake_contraction_value) + wake_contraction_value * (0.887 - 0.125 *
slip_ratio)
'pitch_contraction = (1 - 0.293 * slip_ratio)
If wake_r > 1 Then wake_r = 1

If wake_pitch_set <= 0 Then
'final wake pitch
pw = 0.5 * (advance_ratio + average_P)
Else
'preset value
pw = wake_pitch_set
End If


'increment angle step for wake
dpsi = (2 * pi * wake_length / ((average_P / 2 + 4.5 * pw) / 5)) / no_of_wake_points

For j = 1 To num_sections
 r = propdata(j).radius

 'blade pitch for each section
 'P = propdata(j).pitch
 'P = find_trailing_upper_edge_pitch(j)
 P = find_trailing_edge_pitch(j)
 If P < 0 Then P = 0.01



'set intial pitch of the wake to the propeller pitch
variable_P = P

'first point of wake on trailing edge
wake(j, 1).y = final_section(j, section_data(j).te_id).y
wake(j, 1).x = final_section(j, section_data(j).te_id).x
wake(j, 1).z = final_section(j, section_data(j).te_id).z



For i = 2 To no_of_wake_points


wake(j, i).y = wake(j, i - 1).y + (variable_P * D * dpsi) / (2 * pi)


If wake(j, i).y < transition_length Then
'
'transition polynomial for pitch variation
'same for contraction
xsi = (wake(j, i).y - wake(j, 1).y) / (transition_length - wake(j, 1).y)
If xsi < 0 Then xsi = 0
fun = Sqr(xsi) + 1.013 * xsi - 1.92 * xsi ^ 2 + 1.228 * xsi ^ 3 - 0.321 * xsi ^ 4


variable_P = P - (P - pw) * fun
'variable_P = P - (P - pw) * (wake(j, i).y / transition_length)


Else
variable_P = pw
End If

wake(j, i).x = wake(j, i - 1).x * Cos(-dpsi) - wake(j, i - 1).z * Sin(-dpsi)
wake(j, i).z = wake(j, i - 1).x * Sin(-dpsi) + wake(j, i - 1).z * Cos(-dpsi)


Next


'contract the wake
'*****************************************************************
****
If wake_r < 1 Then
Dim f(7) As Single
f(2) = 0.27
f(3) = 0.5
f(4) = 0.65
f(5) = 0.7
f(6) = 0.87
f(7) = 0.95

For i = 2 To no_of_wake_points
If wake(j, i).y < transition_length Then
'
```

```
'transition polynomial for contraction shape
xsi = (wake(j, i).y - wake(j, 1).y) / (transition_length - wake(j, 1).y)
If xsi < 0 Then xsi = 0
fun = Sqr(xsi) + 1.013 * xsi - 1.92 * xsi ^ 2 + 1.228 * xsi ^ 3 - 0.321 * xsi ^ 4

'contarction value = 1 for full effect and =0 for no contraction
contraction_factor = 1 - (1 - wake_r) * fun
Else
contraction_factor = wake_r
End If

'no contraction on first few points for first section to avoid intersection with hub
'If i = 2 Then contraction_factor = 1
If i > 1 And i < 8 Then
contraction_factor = (1 - (1 - contraction_factor) * f(i))
End If
'smooth transition

wake(j, i).x = wake(j, i).x * contraction_factor
wake(j, i).z = wake(j, i).z * contraction_factor
Next
End If
'end of wake contraction


Next

End Sub
Private Function find_trailing_edge_pitch(j) As Single
Dim average_z As Double
Dim average_y As Double
Dim temp As Double

average_z = (final_section(j, 2).z + final_section(j, section_data(j).num_points - 1).z)
/ 2
average_y = (final_section(j, 2).y + final_section(j, section_data(j).num_points - 1).y)
/ 2

temp = (final_section(j, 1).y - average_y) / (average_z - final_section(j, 1).z)
find_trailing_edge_pitch = temp * 2 * pi * propdata(j).radius / D

End Function
Private Function find_trailing_upper_edge_pitch(j) As Single
Dim average_z As Double
Dim average_y As Double
Dim temp As Double

average_z = final_section(j, section_data(j).num_points - 1).z
average_y = final_section(j, section_data(j).num_points - 1).y
```

```
temp = (final_section(j, 1).y - average_y) / (average_z - final_section(j, 1).z)
find_trailing_upper_edge_pitch = temp * 2 * pi * propdata(j).radius / D

End Function

Private Sub calculate_second_end_of_ring()
'routine that calculates the second section of the ring
Dim i, j, num_edge_points As Integer
Dim angle As Single
num_edge_points = 100
num_edge_section_points = Abs(section_data(num_sections).num_points -
section_data(num_sections).le_id) + 1

angle = 2 * pi / no_of_blades

For i = 1 To section_data(num_sections).le_id
ring_section_edge(10, i).x = Cos(angle) * final_section(num_sections, i).x +
Sin(angle) * final_section(num_sections, i).z
ring_section_edge(10, i).y = final_section(num_sections, i).y
ring_section_edge(10, i).z = -Sin(angle) * final_section(num_sections, i).x +
Cos(angle) * final_section(num_sections, i).z
Next


End Sub
Private Sub calculate_leading_end_of_ring(num_edge_points)
Dim i As Integer
Dim psi, phi, dpsi, r, P As Single

ReDim ring_leading_end(num_edge_points)

'set point to trailing edge of section
ring_leading_end(1).x = final_section(num_sections,
section_data(num_sections).le_id).x
ring_leading_end(1).y = final_section(num_sections,
section_data(num_sections).le_id).y
ring_leading_end(1).z = final_section(num_sections,
section_data(num_sections).le_id).z

r = Sqr(ring_leading_end(1).z ^ 2 + ring_leading_end(1).x ^ 2)

'Calculate phi
P = propdata(num_sections).pitch
If P = 0 Then P = 0.1
phi = Atn(2 * pi * r / P)
psi = section(num_sections, section_data(num_sections).le_id).x / Sqr(r ^ 2 + (P / (2 *
pi)) ^ 2)
dpsi = ((((-ring_width / 2) - ring_leading_end(1).y) / P) * 2 * pi) / (num_edge_points -
1)
psi = psi
```

```
'helical edge
'*********************
For i = 2 To num_edge_points
psi = psi + dpsi
ring_leading_end(i).x = r * Cos(psi)
ring_leading_end(i).y = (P * psi) / (2 * pi)
ring_leading_end(i).z = -r * Sin(psi)
Next

End Sub
Public Sub rans_domain()
Dim i, j As Integer
Dim variable_P, average_P, P As Single
Dim psi, dpsi, r As Single
Dim contraction_factor As Single
Dim start_psi, y_offset As Single
Dim wake_r As Single
Dim xsi As Single
Dim fun As Single
Dim slip_ratio As Single
Dim advance_ratio As Single
Dim n As Single
Dim pw As Single
Dim wake_length As Single


no_of_wake_points = 300
ReDim wake(num_sections, no_of_wake_points)

wake_length = blade_wake_length
transition_length = transition_length * D

'find average_pitch
For j = 1 To num_sections
average_P = average_P + propdata(j).pitch
Next
average_P = average_P / num_sections


'advance speed

advance_ratio = va / (rps * D)
slip_ratio = 1 - advance_ratio / average_P


If wake_pitch_set <= 0 Then
'final wake pitch
pw = 0.5 * (advance_ratio + average_P)
Else
'preset value
```

```
pw = wake_pitch_set
End If


'increment angle step for wake
dpsi = (2 * pi * wake_length / ((average_P / 2 + 4.5 * pw) / 5)) / no_of_wake_points


Dim vector_magnitude As Single

'first point of wake on trailing edge
wake(1, 1) = final_section(1, section_data(1).te_id)
Call rotate_point(wake(1, 1), 60 * pi / 180, 1)

'create starting point for outer helix
'by extending line perpenticular to hub to 2.5 D
wake(2, 1).y = wake(1, 1).y
vector_magnitude = Sqr(wake(1, 1).x ^ 2 + wake(1, 1).z ^ 2)
wake(2, 1).z = 1.25 * D * wake(1, 1).z / vector_magnitude
wake(2, 1).x = 1.25 * D * wake(1, 1).x / vector_magnitude



'create the to helixes
'*****************************************************************
*
 For j = 1 To 2


 'blade pitch for each section
  P = find_trailing_edge_pitch(1)
 If P = 0 Then P = 0.01



'set intial pitch of the wake to the propeller pitch
variable_P = P

If j = 2 Then r = 1.24 * D
If j = 1 Then r = propdata(1).radius

For i = 2 To no_of_wake_points


wake(j, i).y = wake(j, i - 1).y + (variable_P * D * dpsi) / (2 * pi)


If wake(j, i).y < transition_length Then
'

'transition polynomial for pitch variation
```

```
'same for contraction
xsi = (wake(j, i).y - wake(j, 1).y) / (transition_length - wake(j, 1).y)
fun = Sqr(xsi) + 1.013 * xsi - 1.92 * xsi ^ 2 + 1.228 * xsi ^ 3 - 0.321 * xsi ^ 4


variable_P = P - (P - pw) * fun


Else
variable_P = pw
End If

wake(j, i).x = wake(j, i - 1).x * Cos(-dpsi) - wake(j, i - 1).z * Sin(-dpsi)
wake(j, i).z = wake(j, i - 1).x * Sin(-dpsi) + wake(j, i - 1).z * Cos(-dpsi)


Next

Next


'***********************************************************************
******
'Create outer helix in hub area
'***********************************************************************
******
Dim temp(100) As section

'point upstream
wake(3, 1) = final_section(1, section_data(1).le_id)
Call rotate_point(wake(3, 1), 60 * pi / 180, 1)
vector_magnitude = Sqr(wake(3, 1).x ^ 2 + wake(3, 1).z ^ 2)
wake(3, 1).z = 1.25 * D * wake(3, 1).z / vector_magnitude
wake(3, 1).x = 1.25 * D * wake(3, 1).x / vector_magnitude

  Call helix_from_point_to_point(wake(3, 1), wake(2, 1), temp(), 100)
 For j = 1 To 100
   wake(3, j).x = temp(j).x
   wake(3, j).y = temp(j).y
   wake(3, j).z = temp(j).z
 Next



End Sub


Public Sub calculate_final_sections()
Dim i, j As Integer
```

```
ReDim final_section(num_sections, section_data(1).num_points)
For j = 1 To num_sections
 For i = 1 To section_data(j).num_points
  final_section(j, i).x = section(1, i).x * propdata(j).chord
  final_section(j, i).y = section(1, i).y * propdata(j).chord
 Next
Next
End Sub
Public Sub transform_section(section_number, chord, thickness)
Dim i As Integer

'set correct thickness %
'section fixed to 10% in fix sub
For i = 1 To section_data(section_number).num_points
section(section_number, i).y = section(section_number, i).y * (thickness / 0.1)
Next

'scale section to correct chord etc
For i = 1 To section_data(section_number).num_points
section(section_number, i).x = section(section_number, i).x * chord
section(section_number, i).y = section(section_number, i).y * chord
Next

End Sub
Public Sub helical_section(section_number, r, P, blade_rake)
Dim t As Single
Dim phi, psi, start_psi As Single
Dim i As Integer
Dim y_offset As Single
Dim t_total As Single



'Calculate phi
If P = 0 Then P = 0.1
phi = Atn(P * D / (2 * pi * r))
start_psi = propdata(section_number).skew / Sqr(r ^ 2 + (P * D / (2 * pi)) ^ 2)
'y_offset = -(start_psi / (2 * pi)) * P * D


'Map onto helix
'*********************
For i = 1 To section_data(section_number).num_points
psi = start_psi + section(section_number, i).x / Sqr(r ^ 2 + (P * D / (2 * pi)) ^ 2)
t = section(section_number, i).y
final_section(section_number, i).x = r * Cos(psi - t / r * Sin(phi))
final_section(section_number, i).y = y_offset + blade_rake + (P * psi * D) / (2 * pi) +
t * Cos(phi)
final_section(section_number, i).z = -r * Sin(psi - t / r * Sin(phi))
Next
```

```
End Sub
Public Sub hub_arc(hub_length)
Dim radius, f As Single
Dim no_of_points As Integer
Dim start_angle, end_angle As Single

no_of_points = 121


ReDim hub_trailing_edge(no_of_points)
ReDim hub_leading_edge(no_of_points)

radius = Sqr(final_section(1, section_data(1).le_id).x ^ 2 + final_section(1,
section_data(1).le_id).z ^ 2)

'set beginning of arc for hub
f = (hub_trailing_end(1, 100).z / radius) 'position of end on hub

Select Case f
Case Is = -1
If hub_leading_end(1, 100).z > 0 Then start_angle = pi
If hub_leading_end(1, 100).z < 0 Then start_angle = 0
Case Is = 1
If hub_leading_end(1, 100).z > 0 Then start_angle = 0
If hub_leading_end(1, 100).z < 0 Then start_angle = pi
Case Is <> 1, -1
If hub_trailing_end(1, 100).x > 0 Then start_angle = pi / 2 - Atn(f / Sqr(1 - f * f))
If hub_trailing_end(1, 100).x < 0 Then start_angle = 3 * pi / 2 + Atn(f / Sqr(1 - f * f))
End Select
end_angle = start_angle + 2 * pi / no_of_blades
'draws arc
Call arc(hub_trailing_edge(), start_angle, end_angle, radius, 0, hub_length / 2, 0,
no_of_points, 1)


'set beginning of arc for hub
f = (hub_leading_end(1, 100).z / radius) 'position of end on hub

Select Case f
Case Is = -1
If hub_leading_end(1, 100).z > 0 Then start_angle = pi
If hub_leading_end(1, 100).z < 0 Then start_angle = 0
Case Is = 1
If hub_leading_end(1, 100).z > 0 Then start_angle = 0
If hub_leading_end(1, 100).z < 0 Then start_angle = pi
Case Is <> 1, -1
If hub_trailing_end(1, 100).x > 0 Then start_angle = pi / 2 - Atn(f / Sqr(1 - f * f))
If hub_trailing_end(1, 100).x < 0 Then start_angle = 3 * pi / 2 + Atn(f / Sqr(1 - f * f))
```

```
End Select
end_angle = start_angle + 2 * pi / no_of_blades
'draws arc
Call arc(hub_leading_edge(), start_angle, end_angle, radius, 0, -hub_length / 2, 0,
no_of_points, 1)



End Sub

Public Sub hub_arc_near_blade()
Dim radius, f As Single
Dim no_of_points As Integer
Dim start_angle, end_angle As Single

no_of_points = 36


ReDim hub_trailing_arc_connect(no_of_points)
ReDim hub_leading_arc_connect(no_of_points)

radius = Sqr(final_section(1, section_data(1).le_id).x ^ 2 + final_section(1,
section_data(1).le_id).z ^ 2)

'set beginning of arc for hub
f = (hub_trailing_end(1).z / radius) 'position of end on hub

Select Case f
Case Is = -1
If hub_leading_end(1).z > 0 Then start_angle = pi
If hub_leading_end(1).z < 0 Then start_angle = 0
Case Is = 1
If hub_leading_end(1).z > 0 Then start_angle = 0
If hub_leading_end(1).z < 0 Then start_angle = pi
Case Is <> 1, -1
If hub_trailing_end(1).x > 0 Then start_angle = pi / 2 - Atn(f / Sqr(1 - f * f))
If hub_trailing_end(1).x < 0 Then start_angle = 3 * pi / 2 + Atn(f / Sqr(1 - f * f))
End Select
end_angle = start_angle + 2 * pi / no_of_blades
'draws arc
Call arc(hub_trailing_arc_connect(), start_angle, end_angle, radius, 0, hub_length / 2,
0, no_of_points, 1)


'set beginning of arc for hub
f = (hub_leading_end(100).z / radius) 'position of end on hub

Select Case f
Case Is = -1
If hub_leading_end(1).z > 0 Then start_angle = pi
If hub_leading_end(1).z < 0 Then start_angle = 0
Case Is = 1
If hub_leading_end(1).z > 0 Then start_angle = 0
If hub_leading_end(1).z < 0 Then start_angle = pi
Case Is <> 1, -1
If hub_trailing_end(1).x > 0 Then start_angle = pi / 2 - Atn(f / Sqr(1 - f * f))
If hub_trailing_end(1).x < 0 Then start_angle = 3 * pi / 2 + Atn(f / Sqr(1 - f * f))
End Select
end_angle = start_angle + 2 * pi / no_of_blades
'draws arc
Call arc(hub_leading_arc_connect(), start_angle, end_angle, radius, 0, -hub_length /
2, 0, no_of_points, 1)

End Sub
Public Sub fix_section(section_number)
Dim i, j As Integer
Dim min_x, max_x, factor, x As Single
Dim min_y, max_y, max_thick, thick As Single
Dim yup, ydn As Single
Dim max_yup, max_ydn As Single

min_x = 9999999
max_x = -99999999


'Call rotate_section(section_number)


'find min and max x
For i = 1 To section_data(section_number).num_points
If section(section_number, i).x > max_x Then
max_x = section(section_number, i).x
'save trailing edge id
section_data(section_number).te_id = i
End If
If section(section_number, i).x < min_x Then
min_x = section(section_number, i).x
'save leading edge id
section_data(section_number).le_id = i
End If
Next

'set chord length to unit length
'set zero x at mid chord
factor = max_x - min_x
'y_offset = section(section_number, section_data(section_number).te_id).y
For i = 1 To section_data(section_number).num_points
section(section_number, i).x = (section(section_number, i).x - (max_x + min_x) / 2) /
factor
section(section_number, i).y = section(section_number, i).y / factor
```

```
Next

'Open "temp" For Output As 1
'find position of max thickness
max_yup = 0
max_ydn = 0

For x = 0.5 To -0.5 Step -0.001
    i = 0
  Do
    i = i + 1
  Loop Until section(section_number, i).x < x
    j = section_data(section_number).num_points '+ 1
  Do
    j = j - 1
  Loop Until section(section_number, j).x < x

  yup = ((section(section_number, i).y - section(section_number, i - 1).y) /
Abs(section(section_number, i).x - section(section_number, i - 1).x)) *
(section(section_number, i).x - x) + section(section_number, i).y
  ydn = ((section(section_number, j).y - section(section_number, j + 1).y) /
Abs(section(section_number, j).x - section(section_number, j + 1).x)) *
(section(section_number, j).x - x) + section(section_number, j).y


  If yup > max_yup Then max_yup = yup
  If ydn < max_ydn Then max_ydn = ydn
  'Print #1, x, thick, yup, ydn
Next
  thick = Abs(max_yup - max_ydn)
  section_data(section_number).thickness = thick
  section_data(section_number).position = x

'Close #1

'set section to 10% thickness
For i = 1 To section_data(section_number).num_points
section(section_number, i).y = section(section_number, i).y * (0.1 /
section_data(section_number).thickness)
Next

End Sub
Public Sub read_section(section_number As Integer, section_file$)
Dim i As Integer

Open section_file$ For Input As 1
Input #1, section_data(section_number).num_points
For i = 1 To section_data(section_number).num_points
Input #1, section(section_number, i).x, section(section_number, i).y
Next
```

```
Close #1

End Sub
Public Sub read_cap_section(v() As section, section_file$)
Dim i As Integer

Open section_file$ For Input As 1
Input #1, cap_section_points
ReDim v(cap_section_points)
For i = 1 To cap_section_points
Input #1, v(i).y, v(i).x
v(i).z = 0
Next
Close #1

End Sub
Public Sub assign_value(str_dummy$)
Dim group As String
Dim sub_group, property As String


'get first three characters to decide what part it is
'hub,duct etc
group = Left(str_dummy$, 3)

Select Case group
Case "vol"
Call get_value_from_string(str_dummy, volume_mesh)
Case "duc"
'*****************************************************************
*****
'duct
'*****************************************************************
*****
sub_group = Mid(str_dummy$, 6, 2)
'duct images
If sub_group = "im" Then Call get_value_from_string(str_dummy, duct_images)
'duct thickness
If sub_group = "th" Then Call get_value_from_string(str_dummy, duct_thickness)
'panel clustering
If sub_group = "P=" Then Call get_value_from_string(str_dummy, duct_P)
If sub_group = "Q=" Then Call get_value_from_string(str_dummy, duct_Q)

If sub_group = "le" Then
 property = Mid(str_dummy$, 8, 1)
 'length
 If property = "n" Then Call get_value_from_string(str_dummy, duct_length)
 'leading lower duct panels
 If property = "a" Then Call get_value_from_string(str_dummy,
duct_leading_lower_panels_s)
```

```
End If

'duct trailing lower panel s
If sub_group = "tr" Then Call get_value_from_string(str_dummy,
duct_trailing_lower_panels_s)
'duct_panels_nt
If sub_group = "nt" Then Call get_value_from_string(str_dummy, duct_panels_t)
'duct_panels_upper_ns
If sub_group = "up" Then Call get_value_from_string(str_dummy,
duct_upper_panels_s)
'wake properties
If sub_group = "wa" Then
property = Mid(str_dummy$, 11, 6)
'free wake ns
If property = "free_n" Then Call get_value_from_string(str_dummy,
duct_freewake_panels_s)
'fixed wake ns
If property = "fixed_" Then Call get_value_from_string(str_dummy,
duct_fixedwake_panels_s)
'wake length as multiples of chord
If property = "length" Then Call get_value_from_string(str_dummy,
duct_wake_length)
'free wake length as a ratio of fixed wake length
If property = "free_l" Then Call get_value_from_string(str_dummy,
duct_freewake_length)
End If
'*****************************************************************
***
'blade options
'*****************************************************************
***
Case "bla"
'blade options
sub_group = Mid(str_dummy$, 7, 2)
'num of panels
If sub_group = "nt" Then Call get_value_from_string(str_dummy, blade_panels_t)
If sub_group = "ns" Then Call get_value_from_string(str_dummy, blade_panels_s)
If sub_group = "nu" Then Call get_value_from_string(str_dummy, no_of_blades)
'increase panel in the radial direction by this factor
If sub_group = "ti" Then Call get_value_from_string(str_dummy, blade_tip_cluster)
'advance speed
If sub_group = "ad" Then Call get_value_from_string(str_dummy, va)
'rev per second
If sub_group = "rp" Then Call get_value_from_string(str_dummy, rps)
'panel clustering
If sub_group = "P=" Then Call get_value_from_string(str_dummy, blade_P)
If sub_group = "Q=" Then Call get_value_from_string(str_dummy, blade_Q)

'wake properties
If sub_group = "wa" Then
```

```
property = Mid(str_dummy$, 12, 6)
'wake length
If property = "length" Then Call get_value_from_string(str_dummy,
blade_wake_length)
'free wake length as % of wake length
If property = "free_l" Then Call get_value_from_string(str_dummy,
blade_freewake_length)
'fixed wake panels
If property = "fixed_" Then Call get_value_from_string(str_dummy,
blade_fixedwake_panels_s)
'free wake ns
If property = "free_n" Then Call get_value_from_string(str_dummy,
blade_freewake_panels_s)
'wake contraction amount 1 for full 0 for none
If property = "contra" Then Call get_value_from_string(str_dummy,
wake_contraction_value)
'wake transition legth
If property = "transi" Then Call get_value_from_string(str_dummy, transition_length)
'wake final pitch if negative calculated within the program
If property = "final_" Then Call get_value_from_string(str_dummy, wake_pitch_set)

End If
'***********************************************************
'Ring options
'***********************************************************
Case "rin"
sub_group = Mid(str_dummy$, 6, 2)

If sub_group = "sp" Then Call get_value_from_string(str_dummy, ring_split)
If sub_group = "st" Then Call get_value_from_string(str_dummy, no_of_ring_strips)
If sub_group = "nt" Then Call get_value_from_string(str_dummy, ring_panels_t)
If sub_group = "wi" Then Call get_value_from_string(str_dummy, ring_width)
'***********************************************************
'Hub options
'***********************************************************
Case "hub"
sub_group = Mid(str_dummy$, 5, 2)
If sub_group = "st" Then Call get_value_from_string(str_dummy, no_of_hub_strips)
If sub_group = "nt" Then Call get_value_from_string(str_dummy, hub_panels_t)
If sub_group = "ns" Then Call get_value_from_string(str_dummy, hub_panels_s)
If sub_group = "le" Then Call get_value_from_string(str_dummy, hub_length)
If sub_group = "vl" Then Call get_value_from_string(str_dummy,
hub_v_leading_factor)
If sub_group = "vt" Then Call get_value_from_string(str_dummy,
hub_v_trailing_factor)
If sub_group = "of" Then Call get_value_from_string(str_dummy, hub_offset_le)
'***********************************************************
'Cap options
'***********************************************************
Case "cap"
```

```
sub_group = Mid(str_dummy$, 5, 2)
If sub_group = "in" Then Call get_value_from_string(str_dummy, internal_fraction)
If sub_group = "si" Then Call get_value_from_string(str_dummy, side_fraction)
If sub_group = "nt" Then Call get_value_from_string(str_dummy, cap_panels_t)
If sub_group = "ns" Then Call get_value_from_string(str_dummy, cap_panels_s)
If sub_group = "se" Then Call get_value_from_string(str_dummy, cap_auto)
End Select
End Sub
Public Sub get_value_from_string(str_dummy, value)
Dim equal, counter As Integer

counter = 0
equal = 0
Do
counter = counter + 1
If Mid(str_dummy, counter, 1) = "=" Then equal = 1
Loop While equal = 0
value = val(Right(str_dummy, (Len(str_dummy) - counter)))
End Sub
Public Sub read_line(str_dummy)
Dim comment As Integer
Do
comment = 0
Do
Input #7, str_dummy
Loop While str_dummy = ""
If Left(str_dummy, 1) = "!" Then comment = 1
Loop While comment = 1

End Sub

Public Sub read_prop_data()
Dim header As String
Dim str_dummy As String
Dim i As Integer
Dim strdummy As String

Open working_path$ + "propeller.dat" For Input As 1
Input #1, header$
Input #1, str_dummy$
Call get_value_from_string(str_dummy$, num_sections)
Input #1, str_dummy$
Call get_value_from_string(str_dummy$, D)

ReDim propdata(-10 To num_sections)

Input #1, strdummy$
For i = 1 To num_sections
Input #1, propdata(i).radius, propdata(i).chord, propdata(i).skew, propdata(i).rake,
propdata(i).pitch, propdata(i).thickness
propdata(i).radius = propdata(i).radius * D / 2
propdata(i).chord = propdata(i).chord * D
propdata(i).rake = Tan(propdata(i).rake / 57.3) * propdata(i).radius
propdata(i).skew = (propdata(i).skew / 57.3) * propdata(i).radius
Next
Close #1

Open working_path$ + "propoptions.txt" For Input As 7
Do
Call read_line(str_dummy$)
Call assign_value(str_dummy$)
Loop Until EOF(7) = True
Close #7

End Sub
Public Sub spline_through_edges()

Dim leading_points() As section
Dim trailing_points() As section
Dim i As Integer

ReDim leading_points(num_sections)
ReDim trailing_points(num_sections)


ReDim leading_spline(101)
ReDim trailing_spline(101)


For i = 1 To num_sections
leading_points(i).x = final_section(i, section_data(i).le_id).x
leading_points(i).y = final_section(i, section_data(i).le_id).y
leading_points(i).z = final_section(i, section_data(i).le_id).z

trailing_points(i).x = final_section(i, section_data(i).te_id).x
trailing_points(i).y = final_section(i, section_data(i).te_id).y
trailing_points(i).z = final_section(i, section_data(i).te_id).z
Next

Call s_spline(leading_points(), 1, 1, leading_spline, num_sections, 100)
Call s_spline(trailing_points(), 1, 1, trailing_spline, num_sections, 100)

End Sub

Public Sub hub_section_strip()
Dim i As Integer
Dim angle, angle_step As Single

angle_step = (360 / no_of_blades / no_of_hub_strips) / 57.2957795130823
```

```
angle = 0
For i = -1 To -no_of_hub_strips Step -1
angle = angle + angle_step
propdata(i).thickness = propdata(1).thickness * Cos(2 * angle)
Next

End Sub
Public Sub helix_from_blade_leading_edge2()
Dim angle As Single
Dim cosangle As Single
Dim a, b, c, r, P As Single
Dim psi, dpsi, phi, new_phi As Single
Dim y_offset As Single
Dim end_psi As Single

Dim i, num_of_points As Integer

num_of_points = 121

ReDim hub_helix_le(num_of_points)

b = Sqr((hub_leading_end(1, 1).x - hub_leading_end(1, 2).x) ^ 2 +
(hub_leading_end(1, 1).y - hub_leading_end(1, 2).y) ^ 2 + (hub_leading_end(1, 1).z -
hub_leading_end(1, 2).z) ^ 2)
c = Sqr((final_section(1, section_data(1).le_id).x - final_section(1,
section_data(1).le_id + 1).x) ^ 2 + (final_section(1, section_data(1).le_id).y -
final_section(1, section_data(1).le_id + 1).y) ^ 2 + (final_section(1,
section_data(1).le_id).z - final_section(1, section_data(1).le_id + 1).z) ^ 2)
a = Sqr((hub_leading_end(1, 2).x - final_section(1, section_data(1).le_id + 1).x) ^ 2 +
(hub_leading_end(1, 2).y - final_section(1, section_data(1).le_id + 1).y) ^ 2 +
(hub_leading_end(1, 2).z - final_section(1, section_data(1).le_id + 1).z) ^ 2)
cosangle = (b ^ 2 + c ^ 2 - a ^ 2) / (2 * b * c)
angle = Atn(-cosangle / Sqr(-cosangle * cosangle + 1)) + 2 * Atn(1)

'set point to leading edge of first section
hub_helix_le(1).x = final_section(1, section_data(1).le_id).x
hub_helix_le(1).y = final_section(1, section_data(1).le_id).y
hub_helix_le(1).z = final_section(1, section_data(1).le_id).z


r = Sqr(hub_helix_le(1).z ^ 2 + hub_helix_le(1).x ^ 2)


phi = Atn(propdata(1).pitch * D / (2 * pi * r))
new_phi = (phi + angle / 2)

P = Tan(new_phi) * 2 * pi * r + hub_v_leading_factor

If P = 0 Then P = 0.05
'calculate the start angle and the step angle
```

```
'calculate start psi so it matches with leading edge of section
psi = -Atn((final_section(1, section_data(1).le_id).z / r) / Sqr(-(final_section(1,
section_data(1).le_id).z / r) ^ 2 + 1))
end_psi = -(2 * pi / no_of_blades) * (propdata(1).pitch * D) / (P - (propdata(1).pitch *
D))
dpsi = end_psi / (num_of_points - 1)
psi = psi - dpsi
'helical edge
'**********************
For i = 1 To Int(num_of_points / 2)
psi = psi + dpsi
hub_helix_le(i).x = r * Cos(psi)
hub_helix_le(i).y = (P * psi) / (2 * pi)
hub_helix_le(i).z = -r * Sin(psi)
Next

y_offset = final_section(1, section_data(1).le_id).y - hub_helix_le(1).y

'move helix so it matches blade le
For i = 1 To num_of_points
hub_helix_le(i).y = hub_helix_le(i).y + y_offset
Next

Dim temp(70) As section

Call helix_from_point_to_point(hub_helix_le(Int(num_of_points / 2)),
hub_section_edge(10, section_data(1).le_id), temp(), num_of_points -
Int(num_of_points / 2))

For i = Int(num_of_points / 2) + 1 To num_of_points
hub_helix_le(i).x = temp(i - Int(num_of_points / 2)).x
hub_helix_le(i).y = temp(i - Int(num_of_points / 2)).y
hub_helix_le(i).z = temp(i - Int(num_of_points / 2)).z
Next

End Sub
Public Sub ring_helix_from_blade_leading_edge()
Dim angle As Single
Dim cosangle As Single
Dim a, b, c, r, P As Single
Dim psi, dpsi, phi, new_phi As Single
Dim y_offset As Single
Dim end_psi As Single


Dim i, num_of_points As Integer

num_of_points = 121
```

```
ReDim ring_helix_le(num_of_points)

b = Sqr((ring_leading_end(1, 1).x - ring_leading_end(1, 2).x) ^ 2 +
(ring_leading_end(1, 1).y - ring_leading_end(1, 2).y) ^ 2 + (ring_leading_end(1, 1).z
- ring_leading_end(1, 2).z) ^ 2)
c = Sqr((final_section(num_sections, section_data(num_sections).le_id).x -
final_section(num_sections, section_data(num_sections).le_id + 1).x) ^ 2 +
(final_section(num_sections, section_data(num_sections).le_id).y -
final_section(num_sections, section_data(num_sections).le_id + 1).y) ^ 2 +
(final_section(1, section_data(1).le_id).z - final_section(1, section_data(1).le_id +
1).z) ^ 2)
a = Sqr((ring_leading_end(1, 2).x - final_section(num_sections,
section_data(num_sections).le_id + 1).x) ^ 2 + (ring_leading_end(1, 2).y -
final_section(num_sections, section_data(num_sections).le_id + 1).y) ^ 2 +
(ring_leading_end(1, 2).z - final_section(num_sections,
section_data(num_sections).le_id + 1).z) ^ 2)
cosangle = (b ^ 2 + c ^ 2 - a ^ 2) / (2 * b * c)
angle = Atn(-cosangle / Sqr(-cosangle * cosangle + 1)) + 2 * Atn(1)

'set point to leading edge of first section
ring_helix_le(1).x = final_section(num_sections, section_data(num_sections).le_id).x
ring_helix_le(1).y = final_section(num_sections, section_data(num_sections).le_id).y
ring_helix_le(1).z = final_section(num_sections, section_data(num_sections).le_id).z


r = Sqr(ring_helix_le(1).z ^ 2 + ring_helix_le(1).x ^ 2)


phi = Atn(propdata(num_sections).pitch * D / (2 * pi * r))
new_phi = (phi + angle / 2)

P = Tan(new_phi) * 2 * pi * r '+ ring_v_leading_factor

If P = 0 Then P = 0.05
'calculate the start angle and the step angle

'calculate start psi so it matches with leading edge of section
psi = -Atn((final_section(num_sections, section_data(num_sections).le_id).z / r) /
Sqr(-(final_section(num_sections, section_data(num_sections).le_id).z / r) ^ 2 + 1))
end_psi = -(pi / 2) * (propdata(num_sections).pitch * D) / (P -
(propdata(num_sections).pitch * D))
dpsi = end_psi / (num_of_points - 1)
psi = psi - dpsi
'helical edge
'*********************
For i = 1 To Int(num_of_points / 2)
psi = psi + dpsi
ring_helix_le(i).x = r * Cos(psi)
ring_helix_le(i).y = (P * psi) / (2 * pi)
ring_helix_le(i).z = -r * Sin(psi)
```

```
Next

y_offset = final_section(num_sections, section_data(num_sections).le_id).y -
ring_helix_le(1).y

'move helix so it matches blade le
For i = 1 To num_of_points
ring_helix_le(i).y = ring_helix_le(i).y + y_offset
Next

Dim temp(70) As section

Call helix_from_point_to_point(ring_helix_le(Int(num_of_points / 2)),
ring_section_edge(10, section_data(num_sections).le_id), temp(), num_of_points -
Int(num_of_points / 2))

For i = Int(num_of_points / 2) + 1 To num_of_points
ring_helix_le(i).x = temp(i - Int(num_of_points / 2)).x
ring_helix_le(i).y = temp(i - Int(num_of_points / 2)).y
ring_helix_le(i).z = temp(i - Int(num_of_points / 2)).z
Next

End Sub

Public Sub helix_from_point_to_point(point1 As section, point2 As section, store()
As section, num_of_points As Integer)

Dim r, P As Single
Dim psi, dpsi, phi, new_phi As Single
Dim y_offset As Single
Dim end_psi As Single
Dim angle

Dim i As Integer



'calculate radius from first point, about y axis
r = Sqr(point1.z ^ 2 + point1.x ^ 2)
If r = 0 Then r = 0.03
'calculate start psi
If point1.x > 0 And point1.z < 0 Then psi = -asin(point1.z / r)
If point1.x > 0 And point1.z > 0 Then psi = 2 * pi - asin(point1.z / r)
If point1.x < 0 And point1.z > 0 Then psi = 2 * pi - pi / 2 + asin(point1.x / r)
If point1.x < 0 And point1.z < 0 Then psi = pi / 2 - asin(point1.x / r)
'calculate end psi so it matches with trailing edge of section
'If point2.x > 0 Then end_psi = -Atn((point2.z / r) / Sqr(-(point2.z / r) ^ 2 + 1))
'If point2.x < 0 Then end_psi = pi / 2 - Atn((point2.x / r) / Sqr(-(point2.x / r) ^ 2 + 1))
If point2.x > 0 And point2.z < 0 Then end_psi = -asin(point2.z / r)
If point2.x > 0 And point2.z > 0 Then end_psi = 2 * pi - asin(point2.z / r)
If point2.x < 0 And point2.z > 0 Then end_psi = 2 * pi - pi / 2 + asin(point2.x / r)
If point2.x < 0 And point2.z < 0 Then end_psi = pi / 2 - asin(point2.x / r)

'calculate pitch
If psi < end_psi Then angle = end_psi - psi
If psi > end_psi Then angle = (2 * pi - psi) + end_psi
P = 2 * pi * (point2.y - point1.y) / angle

dpsi = angle / (num_of_points - 1)
psi = psi - dpsi

'helical edge
'*********************
For i = 1 To num_of_points
psi = psi + dpsi
store(i).x = r * Cos(psi)
store(i).y = (P * psi) / (2 * pi)
store(i).z = -r * Sin(psi)
Next

y_offset = point1.y - store(1).y

'move helix so it matches blade le
For i = 1 To num_of_points
store(i).y = store(i).y + y_offset
Next

End Sub
Public Sub hub_intermediate_helix()
Dim step, index, i, j As Integer
Dim temp(100) As section

step = 120 / no_of_hub_strips

index = 1
'helixes for front end of hub
For i = 1 To no_of_hub_strips
   index = index + step
   Call helix_from_point_to_point(hub_leading_edge(index), hub_helix_le(index),
temp(), 100)
  For j = 1 To 100
   hub_leading_end(i + 1, j).x = temp(j).x
   hub_leading_end(i + 1, j).y = temp(j).y
   hub_leading_end(i + 1, j).z = temp(j).z
  Next
Next


'helixes for the rear face
index = -step
```

```
For i = 1 To no_of_hub_strips + 1
   index = index + step
   Call helix_from_point_to_point(hub_helix_te(index + 1), hub_trailing_edge(121 -
index), temp(), 100)
  For j = 1 To 100
   hub_trailing_end(i, j).x = temp(j).x
   hub_trailing_end(i, j).y = temp(j).y
   hub_trailing_end(i, j).z = temp(j).z
  Next
Next

'helixes between faces
index = -step

ReDim hub_helix_blades(10, 100)

For i = 1 To no_of_hub_strips + 1
   index = index + step
   Call helix_from_point_to_point(hub_helix_le(index + 1), hub_helix_te(121 -
index), temp(), 100)
  For j = 1 To 100
   hub_helix_blades(i, j).x = temp(j).x
   hub_helix_blades(i, j).y = temp(j).y
   hub_helix_blades(i, j).z = temp(j).z
  Next
Next



End Sub
Public Sub ring_intermediate_helix()
Dim step, index, i, j As Integer
Dim temp(100) As section

step = 120 / no_of_ring_strips

'helixes for front end of ring
For i = 1 To no_of_ring_strips
   index = index + step
   Call helix_from_point_to_point(ring_leading_edge(index), ring_helix_le(index),
temp(), 100)
  For j = 1 To 100
   ring_leading_end(i + 1, j).x = temp(j).x
   ring_leading_end(i + 1, j).y = temp(j).y
   ring_leading_end(i + 1, j).z = temp(j).z
  Next
Next
```

```
'helixes for the rear face
index = -step

For i = 1 To no_of_ring_strips + 1
   index = index + step
   Call helix_from_point_to_point(ring_helix_te(index + 1), ring_trailing_edge(121 -
index), temp(), 100)
  For j = 1 To 100
   ring_trailing_end(i, j).x = temp(j).x
   ring_trailing_end(i, j).y = temp(j).y
   ring_trailing_end(i, j).z = temp(j).z
  Next
Next

'helixes between faces
index = -step

ReDim ring_helix_blades(10, 100)

For i = 1 To no_of_ring_strips + 1
   index = index + step
   Call helix_from_point_to_point(ring_helix_le(index + 1), ring_helix_te(121 -
index), temp(), 100)
  For j = 1 To 100
   ring_helix_blades(i, j).x = temp(j).x
   ring_helix_blades(i, j).y = temp(j).y
   ring_helix_blades(i, j).z = temp(j).z
  Next
Next



End Sub

Public Sub helix_from_blade_trailing_edge2()
Dim angle As Single
Dim cosangle As Single
Dim a, b, c, r, P As Single
Dim psi, dpsi, phi, new_phi As Single
Dim y_offset As Single
Dim end_psi As Single


Dim i, num_of_points As Integer
num_of_points = 121

ReDim hub_helix_te(num_of_points)
```

```
b = Sqr((hub_trailing_end(1, 1).x - hub_trailing_end(1, 2).x) ^ 2 +
(hub_trailing_end(1, 1).y - hub_trailing_end(1, 2).y) ^ 2 + (hub_trailing_end(1, 1).z -
hub_trailing_end(1, 2).z) ^ 2)
c = Sqr((final_section(1, section_data(1).te_id).x - final_section(1,
section_data(1).te_id + 1).x) ^ 2 + (final_section(1, section_data(1).te_id).y +
final_section(1, section_data(1).te_id + 1).y) ^ 2 + (final_section(1,
section_data(1).le_id).z - final_section(1, section_data(1).le_id + 1).z) ^ 2)
a = Sqr((hub_trailing_end(1, 2).x - final_section(1, section_data(1).te_id + 1).x) ^ 2 +
(hub_trailing_end(1, 2).y + final_section(1, section_data(1).te_id + 1).y) ^ 2 +
(hub_trailing_end(1, 2).z - final_section(1, section_data(1).te_id + 1).z) ^ 2)

cosangle = (b ^ 2 + c ^ 2 - a ^ 2) / (2 * b * c)
If cosangle < 0 Then cosangle = 0
If cosangle > 1 Then cosangle = 0.95
angle = Atn(-cosangle / Sqr(-cosangle * cosangle + 1)) + 2 * Atn(1)

'set point to trailing edge of first section
hub_helix_te(1).x = hub_section_edge(10, 1).x
hub_helix_te(1).y = hub_section_edge(10, 1).y
hub_helix_te(1).z = hub_section_edge(10, 1).z


r = Sqr(hub_helix_te(1).z ^ 2 + hub_helix_te(1).x ^ 2)

phi = Atn(propdata(1).pitch / (2 * pi * r))
new_phi = (phi + angle / 2)

P = Tan(new_phi) * 2 * pi * r + hub_v_trailing_factor

If P = 0 Then P = 0.05
'calculate the start angle and the step angle

'calculate start psi so it matches with trailing edge of section
If hub_section_edge(10, 1).x > 0 Then psi = -Atn((hub_section_edge(10, 1).z / r) /
Sqr(-(hub_section_edge(10, 1).z / r) ^ 2 + 1))
If hub_section_edge(10, 1).x < 0 Then psi = pi / 2 - Atn((hub_section_edge(10, 1).x /
r) / Sqr(-(hub_section_edge(10, 1).x / r) ^ 2 + 1))
'step dpsi for the set number of steps
end_psi = (2 * pi / no_of_blades) * propdata(1).pitch / (P - propdata(1).pitch)
dpsi = end_psi / (num_of_points - 1)
psi = psi - dpsi

'helical edge
'**********************
For i = 1 To Int(num_of_points / 2)
psi = psi + dpsi
hub_helix_te(i).x = r * Cos(psi)
hub_helix_te(i).y = (P * psi) / (2 * pi)
hub_helix_te(i).z = -r * Sin(psi)
Next
```

```
y_offset = hub_section_edge(10, 1).y - hub_helix_te(1).y

'move helix so it matches blade le
For i = 1 To num_of_points
hub_helix_te(i).y = hub_helix_te(i).y + y_offset
Next

Dim temp(70) As section

Call helix_from_point_to_point(final_section(1, section_data(1).te_id),
hub_helix_te(Int(num_of_points / 2)), temp(), num_of_points - Int(num_of_points /
2))

For i = Int(num_of_points / 2) + 1 To num_of_points
hub_helix_te(i).x = temp(num_of_points - i + 1).x
hub_helix_te(i).y = temp(num_of_points - i + 1).y
hub_helix_te(i).z = temp(num_of_points - i + 1).z
Next

End Sub
Public Sub helix_from_blade_leading_edge()
Dim angle As Single
Dim cosangle As Single
Dim ratio, P As Single
Dim psi, dpsi, phi, new_phi As Single
Dim y_offset As Single
Dim end_psi As Single


Dim i, num_of_points As Integer
num_of_points = 121

ReDim hub_helix_le(num_of_points)

'set point to leading edge of first section
hub_helix_le(1).x = final_section(1, section_data(1).le_id).x
hub_helix_le(1).y = final_section(1, section_data(1).le_id).y
hub_helix_le(1).z = final_section(1, section_data(1).le_id).z

ratio = 2
P = -ratio * no_of_blades * hub_v_leading_factor * (hub_leading_end(1, 1).y -
hub_leading_end(1, 100).y) / D
If P = 0 Then P = -0.001


dpsi = (2 * pi / (ratio * no_of_blades)) / (num_of_points / ratio - 1)

'helical edge
'**********************
```

```
For i = 2 To Int(num_of_points / ratio)
hub_helix_le(i).x = hub_helix_le(i - 1).x * Cos(-dpsi) - hub_helix_le(i - 1).z * Sin(-
dpsi)
hub_helix_le(i).y = hub_helix_le(i - 1).y + (P * D * dpsi) / (2 * pi)
hub_helix_le(i).z = hub_helix_le(i - 1).x * Sin(-dpsi) + hub_helix_le(i - 1).z * Cos(-
dpsi)
Next

Dim temp(121) As section

Call helix_from_point_to_point(hub_helix_le(Int(num_of_points / ratio)),
hub_section_edge(10, section_data(1).le_id), temp(), num_of_points -
Int(num_of_points / ratio))

Dim i_init As Integer

i_init = Int(num_of_points / ratio)
For i = Int(num_of_points / ratio) + 1 To num_of_points

hub_helix_le(i).x = temp(i - i_init).x
hub_helix_le(i).y = temp(i - i_init).y
hub_helix_le(i).z = temp(i - i_init).z
Next

End Sub
Public Sub helix_from_blade_trailing_edge()
Dim angle As Single
Dim cosangle As Single
Dim ratio, P As Single
Dim psi, dpsi, phi, new_phi As Single
Dim y_offset As Single
Dim end_psi As Single


Dim i, num_of_points As Integer
num_of_points = 121

ReDim hub_helix_te(num_of_points)

'set point to trailing edge of first section
hub_helix_te(1).x = hub_section_edge(10, 1).x
hub_helix_te(1).y = hub_section_edge(10, 1).y
hub_helix_te(1).z = hub_section_edge(10, 1).z

ratio = 2
P = ratio * no_of_blades * hub_v_trailing_factor * (hub_trailing_end(1, 1).y -
hub_trailing_end(1, 100).y) / D
If P = 0 Then P = 0.001
```

```
dpsi = -(2 * pi / (ratio * no_of_blades)) / (num_of_points / ratio - 1)

'helical edge
'*********************
For i = 2 To Int(num_of_points / ratio)
hub_helix_te(i).x = hub_helix_te(i - 1).x * Cos(-dpsi) - hub_helix_te(i - 1).z * Sin(-
dpsi)
hub_helix_te(i).y = hub_helix_te(i - 1).y + (P * D * dpsi) / (2 * pi)
hub_helix_te(i).z = hub_helix_te(i - 1).x * Sin(-dpsi) + hub_helix_te(i - 1).z * Cos(-
dpsi)
Next

Dim temp(121) As section

Call helix_from_point_to_point(final_section(1, section_data(1).te_id),
hub_helix_te(Int(num_of_points / ratio)), temp(), num_of_points - Int(num_of_points
/ ratio))

For i = Int(num_of_points / ratio) + 1 To num_of_points
hub_helix_te(i).x = temp(num_of_points - i + 1).x
hub_helix_te(i).y = temp(num_of_points - i + 1).y
hub_helix_te(i).z = temp(num_of_points - i + 1).z
Next

End Sub

Public Sub ring_helix_from_blade_trailing_edge()
Dim angle As Single
Dim cosangle As Single
Dim a, b, c, r, P As Single
Dim psi, dpsi, phi, new_phi As Single
Dim y_offset As Single
Dim end_psi As Single


Dim i, num_of_points As Integer
num_of_points = 121

ReDim ring_helix_te(num_of_points)

b = Sqr((ring_trailing_end(1, 1).x - ring_trailing_end(1, 2).x) ^ 2 +
(ring_trailing_end(1, 1).y - ring_trailing_end(1, 2).y) ^ 2 + (ring_trailing_end(1, 1).z -
ring_trailing_end(1, 2).z) ^ 2)
c = Sqr((final_section(num_sections, section_data(num_sections).te_id).x -
final_section(num_sections, section_data(num_sections).te_id + 1).x) ^ 2 +
(final_section(num_sections, section_data(num_sections).te_id).y +
final_section(num_sections, section_data(num_sections).te_id + 1).y) ^ 2 +
(final_section(1, section_data(1).le_id).z - final_section(1, section_data(1).le_id +
1).z) ^ 2)
```

```
a = Sqr((ring_trailing_end(1, 2).x - final_section(num_sections,
section_data(num_sections).te_id + 1).x) ^ 2 + (ring_trailing_end(1, 2).y +
final_section(num_sections, section_data(num_sections).te_id + 1).y) ^ 2 +
(ring_trailing_end(1, 2).z - final_section(num_sections,
section_data(num_sections).te_id + 1).z) ^ 2)

cosangle = (b ^ 2 + c ^ 2 - a ^ 2) / (2 * b * c)
angle = Atn(-cosangle / Sqr(-cosangle * cosangle + 1)) + 2 * Atn(1)

'set point to trailing edge of first section
ring_helix_te(1).x = ring_section_edge(10, 1).x
ring_helix_te(1).y = ring_section_edge(10, 1).y
ring_helix_te(1).z = ring_section_edge(10, 1).z


r = Sqr(ring_helix_te(1).z ^ 2 + ring_helix_te(1).x ^ 2)

phi = Atn(propdata(num_sections).pitch / (2 * pi * r))
new_phi = (phi + angle / 2)

P = Tan(new_phi) * 2 * pi * r '+ ring_v_trailing_factor

If P = 0 Then P = 0.05
'calculate the start angle and the step angle

'calculate start psi so it matches with trailing edge of section
If ring_section_edge(10, 1).x > 0 Then psi = -Atn((ring_section_edge(10, 1).z / r) /
Sqr(-(ring_section_edge(10, 1).z / r) ^ 2 + 1))
If ring_section_edge(10, 1).x < 0 Then psi = pi / 2 - Atn((ring_section_edge(10, 1).x /
r) / Sqr(-(ring_section_edge(10, 1).x / r) ^ 2 + 1))
'step dpsi for the set number of steps
'dpsi = ((((-1 / 2) - ring_leading_end(1, 1).y) / P) * 2 * pi) / (100 - 1)
end_psi = (pi / 2) * propdata(num_sections).pitch / (P - propdata(num_sections).pitch)
dpsi = end_psi / (num_of_points - 1)
psi = psi - dpsi

'helical edge
'*********************
For i = 1 To Int(num_of_points / 2)
psi = psi + dpsi
ring_helix_te(i).x = r * Cos(psi)
ring_helix_te(i).y = (P * psi) / (2 * pi)
ring_helix_te(i).z = -r * Sin(psi)
Next

y_offset = ring_section_edge(10, 1).y - ring_helix_te(1).y

'move helix so it matches blade le
For i = 1 To num_of_points
ring_helix_te(i).y = ring_helix_te(i).y + y_offset
```

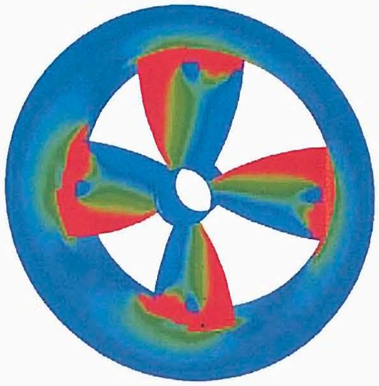
```
Next

Dim temp(70) As section

Call helix_from_point_to_point(final_section(num_sections,
section_data(num_sections).te_id), ring_helix_te(Int(num_of_points / 2)), temp(),
num_of_points - Int(num_of_points / 2))

For i = Int(num_of_points / 2) + 1 To num_of_points
ring_helix_te(i).x = temp(num_of_points - i + 1).x
ring_helix_te(i).y = temp(num_of_points - i + 1).y
ring_helix_te(i).z = temp(num_of_points - i + 1).z
Next

End Sub
```

12 Appendix B

Design Optimisation of a bi-directional integrated thruster

Design optimisation of a bi-directional integrated thruster

Pashias C, Turnock S.R., Abu Sharkh SM.
University of Southampton, UK


Integrated thruster model

ABSTRACT

The majority of thrusters used for the position control of tethered underwater vehicles have asymmetric propulsion characteristics. This paper presents the results of a hydrodynamic design optimisation of a bi-directional integrated thruster. A surface panel method using the perturbation potential method of Morino was used for the optimisation. The model was validated with experimental data giving good agreement. Two versions of the prototype thruster have now been built and tested. In this paper details are given of the design optimisation process for the next generation of thruster for use on a work class Remotely Operated Vehicle. A 2-D potential code coupled with integral boundary layer equations has been used to derive an optimum blade section shape for equal performance in both directions. Using the derived sections the complete thruster was optimised for a given operating condition.

NOMENCLATURE

C_P	Pressure coefficient $= 1 - \left(\dfrac{V_t}{V_R}\right)^2$
D	Diameter
J	Advance ratio $= \dfrac{V}{nD}$
K_T	Thrust coefficient $= \dfrac{T}{\rho n^2 D^4}$
K_Q	Torque coefficient $= \dfrac{Q}{\rho n^2 D^5}$
n	Rotation speed rps
P	Section pitch
Q	Torque
T	Thrust
V	Advance velocity
V_R	$= \sqrt{V^2 + (2\pi nR)^2}$
V_t	Total edge velocity
η	Efficiency $= \dfrac{J}{2\pi}\dfrac{K_T}{K_Q}$

INTRODUCTION

A hydrodynamic design optimisation of a bi-directional thruster is presented. This on-going project (Hughes 2000) has been developed for position control in Remotely Operated Vehicles (ROV), to replace the current hydraulic thrusters. Typically each ROV has six thrusters: four for lateral and two for vertical position control. Most current thrusters have asymmetric propulsion characteristics, because of either off the shelf propellers or motor/shaft blockage effects.

A bi-directional thruster has many advantages over the current hydraulic thrusters. Bi-directionality simplifies the control problem since the same thrust is produced for the forward and reverse condition with the same rpm.

In addition an electric thruster has a lower number of parts, thus reducing maintenance costs. The electric thruster is also lighter which translates into more weight saving by reducing the syntactic foam required for buoyancy. Also no hydraulic fluid has to be pumped down the umbilical cord that can be more than 3000m long, reducing the weight of the cord which means lighter handling gear on the mother vessel.

The first phase of this project was to integrate the permanent magnet (PM) motor with the hydrodynamics of the thruster and to test the concept. Two prototype thrusters were built and tested. Phase one has been completed but did not concentrate on the hydrodynamic aspects of the thruster design. A standard duct and section shape were modified to give bi-directionality, which does not give optimum performance. The second phase of the project is to optimise the hydrodynamic performance of the integrated thruster.

The hydrodynamic analysis of the thruster is carried out using a surface panel code. A mesh generation tool was developed to allow quick definition of arbitrary propellers. The code was validated against standard propellers and experimental data. New section shapes were developed for bi-directional performance. The complete thruster was then modelled and optimised for a given operating condition to match the motor specifications.

INTEGRATED THRUSTER DESIGN

The thruster is powered by a PM motor. The PM ring is attached to the propeller tip and the stator is integrated into the duct. The propeller is driven from the tips and the thrust supported by bearings on the shaft. The bearings are supported by stators from the duct.

Several bearing arrangements were considered (Figure 1), including the thrust bearings supporting the ring, which eliminates the requirement for a hub. The chosen arrangement enables thinner sections to be used offering improved performance. Since the thrust is supported at the hub and the torque at the tip, the twist of the blade helps support the forces.

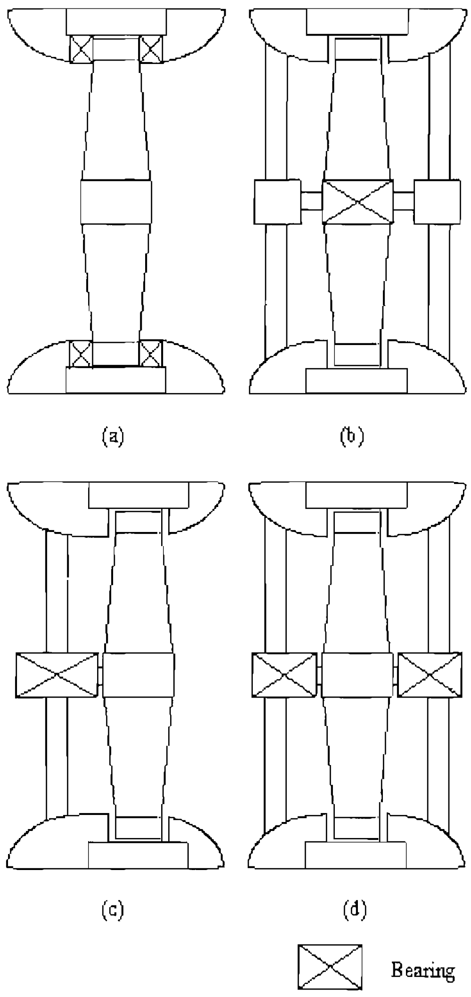


Figure 1 Possible bearing arrangements for integrated thruster

THEORY

Ducted propellers have been widely used in applications where propeller diameter is limited. It is known that ducted propellers are more efficient than open water propellers under such operating conditions (McMahon 1994). The presence of the duct enables the propeller tip to sustain the pressure differential between the back and the face, thus generating more thrust and that is the reason why ducted propeller have larger chords near the tip than open water propellers.

There is a strong interaction between the duct and the rotor and because of the complex nature of the problem a non-viscous lifting surface panel method was used. Such methods can model complex problems quickly and have been used successfully in the past. The ease and time advantage over RANS codes (Turnock 2000) makes them ideal for optimisation studies.

The in house parallel lifting surface panel code, Palisupan (Turnock 1997), was originally developed to solve rudder-propeller interaction and follows the work of Morino (Morino 1974), Newman (Newman 1986) and Lee (Lee 1987). It involves a straightforward application of this method to model the interaction between a rotating propeller and duct.

Laplace's equation can be written as an integral over the bounding surface *S* of a source distribution per unit area *s* and a normal dipole distribution per unit area m distributed over the *S*. This can be expressed in terms of a surface integral as:

$$\phi = \int\int_{S_B} [\tfrac{1}{r}\sigma + \tfrac{\partial}{\partial n}(\tfrac{1}{r})\mu] dS + \int\int_{S_W} \tfrac{\partial}{\partial n}(\tfrac{1}{r})\mu dS \qquad (1)$$

where S_B is the surface of the body and S_W a trailing wake sheet. In the expression *r* is the distance from the point for which the potential is being determined to the integration point on the surface and $\partial/\partial n$ is a partial derivative in the direction normal to the local surface.

Equation (1) is discretised to give the following formulation for the potential at the centre of a given panel:

$$\phi_i = \frac{1}{2\pi}\sum_{j=1}^{N}((\mathbf{U}_\infty \cdot \mathbf{n}_j)S_{ij} - \phi_j D_{ij}) + \sum_{k=1}^{M}\Delta\phi_k W_{ik} \qquad (2)$$

For solving complex flows with multiple bodies the Interaction Velocity Field (IVF) method (Turnock 1994) is used, where the disturbance velocity field generated by a body is superimposed on the velocity field existing in the absence of the body.

For a duct/propeller problem an iterative process is employed as follows:

Step 1. The propeller is solved in the free stream velocity field.
Step 2. The propeller's disturbance velocity is applied to the free stream velocity field and the duct solved.
Step 3. The duct's disturbance velocity is applied to the free stream velocity field of the propeller and solved.
Step 4. Repeat steps 2 and 3 until the solution has converged.

Typically six iterations are required to converge within 0.5% of the total forces. The method effectively splits up the problem to smaller blocks reducing memory requirements and processing time. A typical run takes less than 15min on a Pentium III 1Ghz.

For the thruster to be bi-directional a 180° rotationally symmetric section shape is required. Standard sections are not readily available and a new section had to be developed. A 2-D potential code coupled with integral boundary layer equations as implemented in X-Foil (Drela 1989) was used because of its speed, ease of use and reliability (Milgram 1997).

GEOMETRY AND MESH GENERATION

In order to facilitate the optimisation process a program has been developed to generate the propeller, hub, end caps and duct geometry from standard propeller tables. The geometry is constructed from four sided faces (Figure 2) and exported to the mesh generation tool.

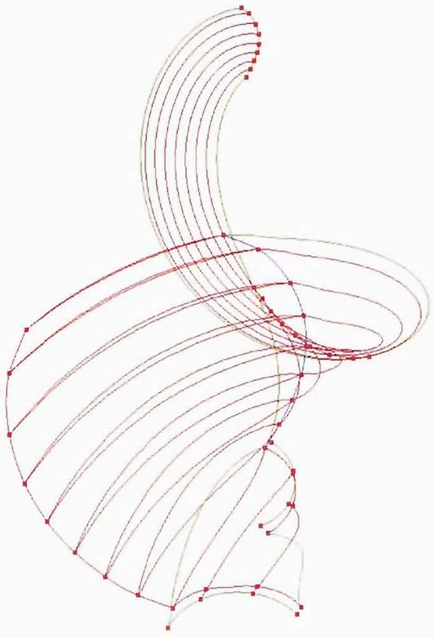


Figure 2 Perspective view of face structure for geometry definition of hub, blade and wake

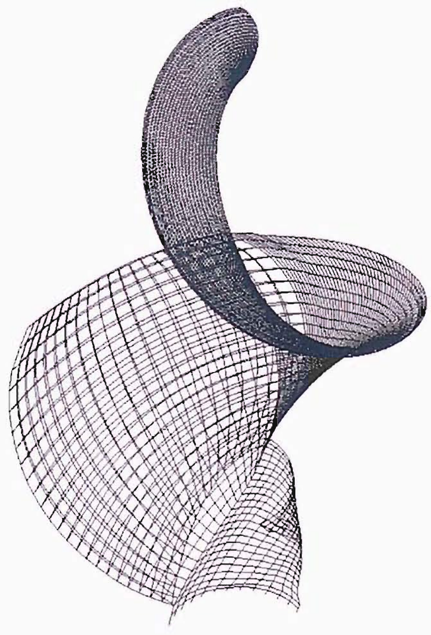


Figure 3 Perspective view of panelling for DMTB4119 propeller and hub

The faces can then be discretised into the desired number of panels (Figure 3) using transfinite interpolation (Hall 1973). Since it is a steady flow problem only one blade of the propeller is generated and the image influence coefficients are used. This again reduces memory and processing requirements enabling more panels to be used. The whole process can be automated enabling variants to be created easily for optimisation studies.

Wake

The wake model is crucial for correct results. Wake relaxation methods have many numerical difficulties and are often unreliable. Because of the tip vortex the wake near the tip of the blade rolls back on itself creating problems (Caponnetto 1997). In addition highly skewed panels can result causing numerical problems. Getting the relaxation to converge is another issue, even when damping is applied (Hughes 1998). A fixed wake model has been chosen because it is robust and quick. Fixed wake models have been used in the past giving as good results without any problems as shown in the 22nd ITTC Propulsion Committee workshop (Gindroz 1998).

The wake bisects the trailing edge of the blade and smoothly varies from that initial pitch to the final wake pitch. Since the experimental data from the two prototype thrusters were available, the final wake pitch was varied until there was good agreement.

The wake transitions from the initial to the final pitch in one propeller diameter. Instead of using a linear transition a 4th order polynomial was used which gives a smooth wakes shape. The wake pitch depends on the wake contraction and the polynomial is the same as the one used for the wake contraction (Hoshino 1991). A wake sensitivity study was carried out and a wake length of four diameters was found to be more than adequate with extra length only changing the thrust and torque by less than 1%.

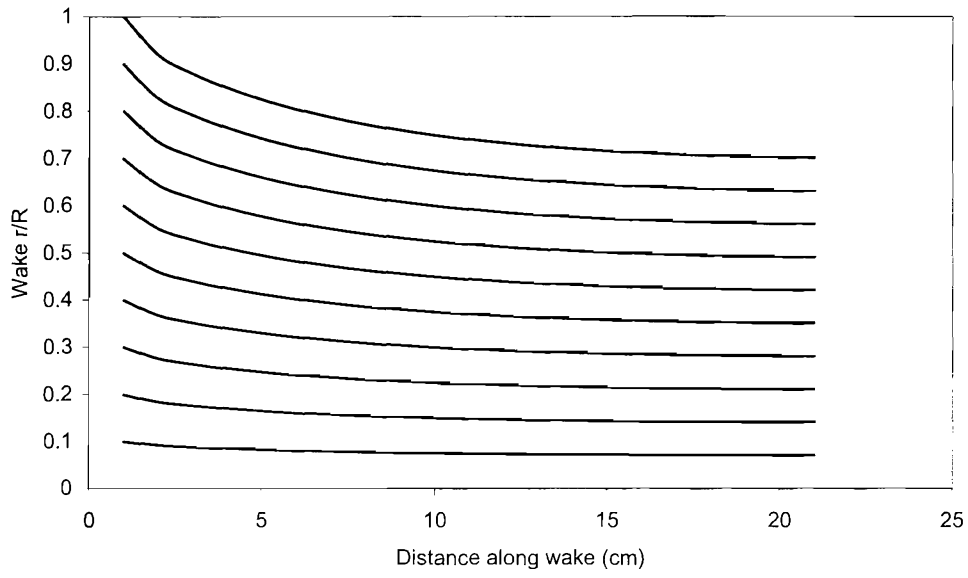


Figure 4 Contraction shape for wake (Hoshino 1991)

Wake contraction has been accounted for by setting it to a fixed value. A final wake contraction is assumed depending on the advance ratio. The wake contraction is set to this value for distances more than one diameter away from the blade. Between the blade and one diameter the wake contraction is modelled by a polynomial, which is based on experimental results (Hoshino 1991). Wake contraction increases with increased propeller loading and from experimental results is about 0.7D to 0.8D for most propellers (Hoshino1991, Pereira 2002).

VALIDATION

To validate the mesh generation and the numerical model, a standard propeller was selected. The DTMB4119 is standard open water three bladed propeller that has been used in the past for validation purposes and experimental data are readily available (Jessup 1998). This propeller has been used for the recent 22nd ITTC Propulsion Committee workshop (Gindroz 1998).

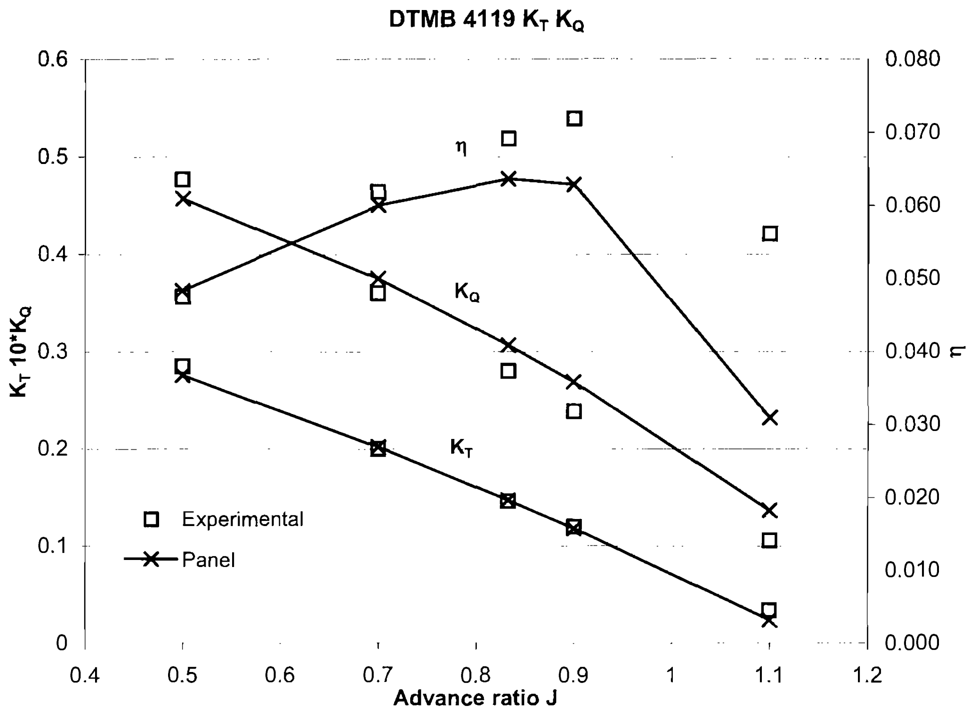
For the DTMB4119 a panel sensitivity study was carried out and 23 panels were used for the blade in the spanwise direction and 20 in the chordwise direction giving a total of 460 panels on the blade with an additional 288 on the hub and 5920 wake panels.

Different wake parameters have been investigated to find their influence on the results. Wake contraction and initial pitch have both been studied.

The initial pitch of the wake has been set to three different values: the local section pitch, the pitch of the bisector of the trailing edge and the pitch of the back face on the trailing edge. The influence has been found to be small giving a change in K_T of 1% and K_Q 0.2% from one extreme to the other. The wake geometry used has the initial pitch set to bisect the trailing edge.

The effect of wake contraction was found to have less than one per cent influence on the K_T and K_Q. For advance ratios close to one the effect was negligible whereas for smaller advance ratios there was a one per cent increase in K_T and K_Q rising with decreasing advance ratio as expected. As the influence was relatively small the wake contraction was not modelled in subsequent calculations.

The pressure distribution for an advance ratio of 0.833 was compared to the experimental data at r/R of 0.3, 0.7 and 0.9. The results were in good agreement with the experimental results and other panel code calculations. The pressures for the 0.9 radius are slightly over predicted, which was the norm for other codes (Hoshino 1998). In addition the K_T, K_Q for a range of J was compared, giving good agreement with the experimental data.



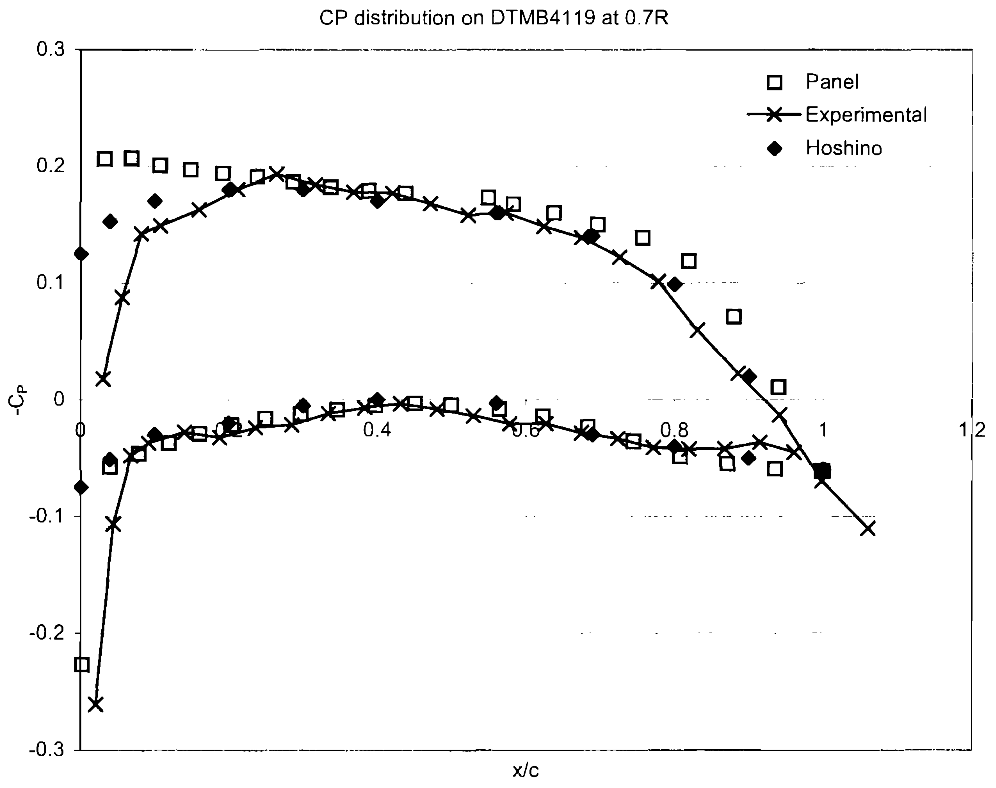


Figure 5 Validation data for the DTMB4119

The numerical model of the duct and propeller was validated using experimental data from the two prototype thrusters. The thrusters were tested with different propeller/duct combinations and had symmetrical ducts and propellers with a P/D of 1.4 and 1.0.

A gap of 1% of the overall propeller diameter was left between the inner surface of the duct and the blade tip to eliminate the high C_P caused by the proximity of the blade tip to shielded panels on the duct. The numerical results were in relatively good agreement with experimental data.

The slope of the K_T and K_Q curves do not match exactly with the experimental results. This is due to a number of reasons. The duct imposes a velocity and hence modifying the operating condition of the propeller. Since no form of wake relaxation was used this was not taken into account. Also ensuring the correct wake shape for heavily loaded propellers is difficult (Takinaci 2001). Another effect not taken into account is that the propeller contraction also affects the duct wake, which will again alter the thrust of the duct.

In addition in this model no viscous interaction effects are included and for simplicity the six stators are neglected. However, as shown previously (Hughes 2000, Hughes 2001) the relative performance changes are captured well.

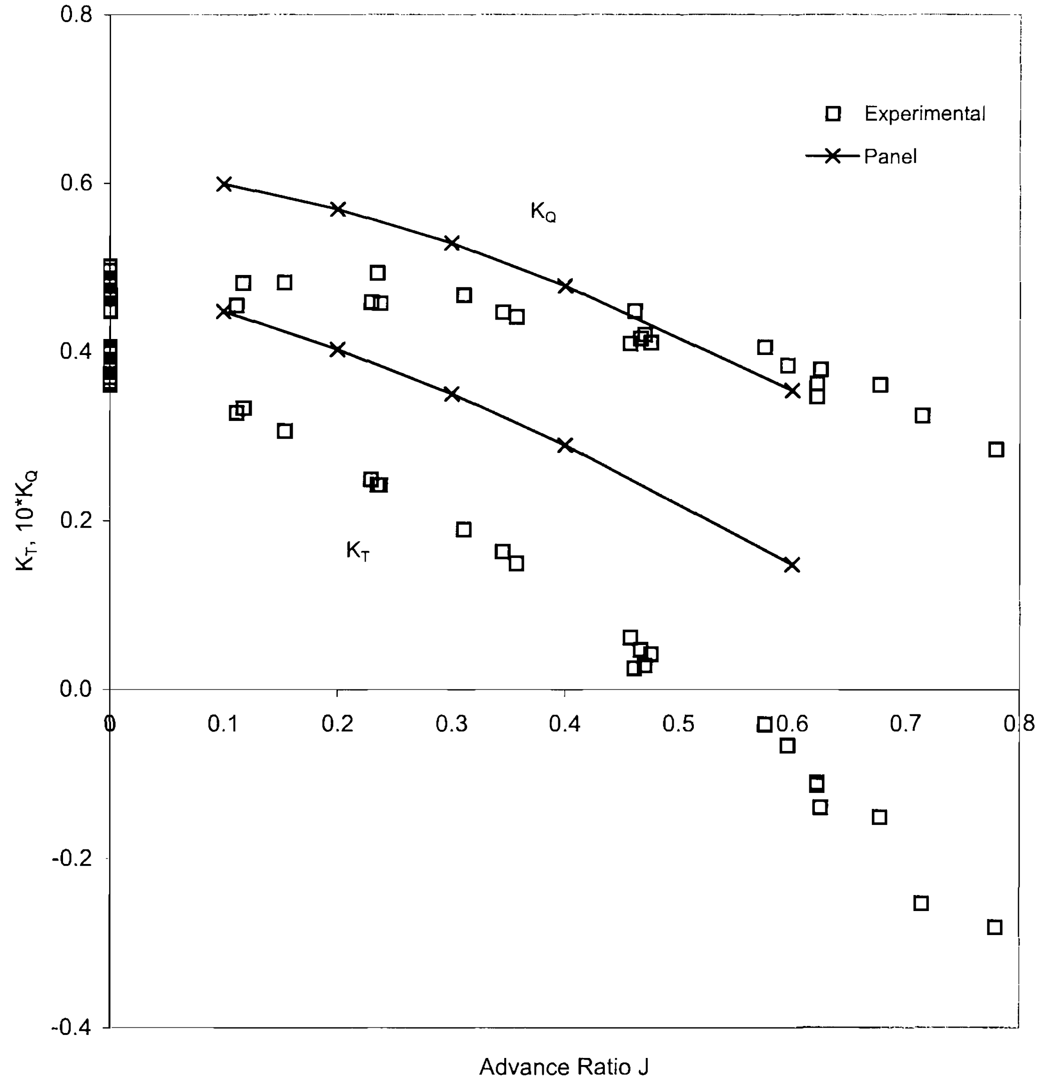


Figure 6 Prototype K_T K_Q

EFFECT OF THE RING

The nature of the design of the integrated thruster includes a ring to which the magnets are attached on the outer surface. An investigation was carried out as to whether it was necessary to include this ring as a rotating surface in the panel method.

The ring was modelled by a cylindrical ring of panels aligned with the pitch at the blade tip, connecting the blades (Figure 7). In previous work (Brown 1994), a similar ring attached to a wake was used to simulate the effect of the duct on the propeller, as a simplification for preliminary design. The effect of the ring was to seal the blade tips and increase the thrust.

Since the ring in this case was in the duct no wake was attached to it. The propeller and ring combination were run in isolation and the results were similar to (Brown 1994). The ringed propeller was then solved in combination with a duct with a cut out ring (Figure 8).

The effect of the ring has so far been to make a non-physically large increase in thrust and also torque. There is also an increase in thrust from the duct. The friction on the ring results in an increase in torque, which offsets the gain in thrust and the increase in efficiency is negligible (Figure 9). The torque due to the friction of the ring is approximately 7% of the overall torque. The ring width was kept to a minimum for the final design to keep frictional drag as low as possible.

Other investigations have been carried out to study the turbulent Taylor-Couette flow in the gap between the ring and the motor windings (Batten 2002). This contribution to torque is not included in the external hydrodynamic optimisation as there is little interaction between the two regimes.

For the optimisation studies presented the effect of the ring is not included and a constant tip gap of 1% of D is used. This is comparable in size to a typical panel dimension on the duct and propeller.

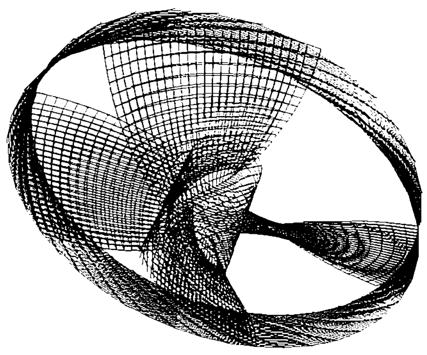


Figure 7 Propeller with ring

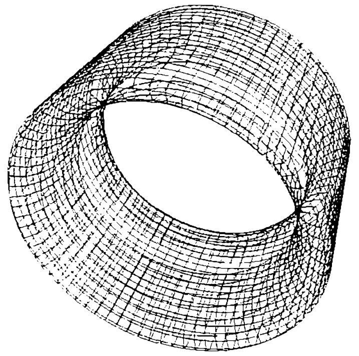


Figure 8 Duct with cut out for ring

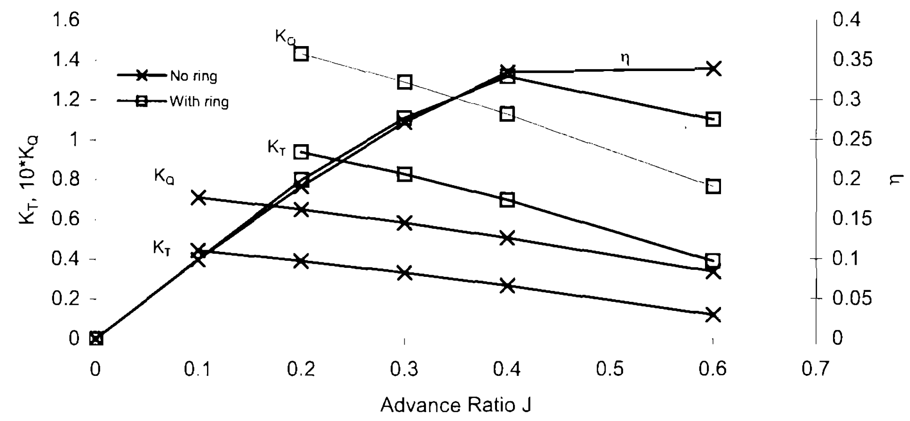


Figure 9 Effect of modelling the ring of the thruster

OPTIMISATION

The thruster was optimised to give maximum bollard pull at zero advance speed with the given motor characteristics. There is an optimum motor rpm, which depends on the viscous losses due to the friction in the gap between the rotor ring and the stator (Batten 2002). As the rpm increase the motor efficiency increases but so do the gap losses, so an optimum exists.

The typical operating depth of the thruster will be from 300m to 3000m. At those depths cavitation is not an issue since the pressure is very high. Small blade area ratios (BAR) can be used with heavily loaded blades without cavitation problems.

To speed up the optimisation process the stators were neglected from the numerical model. The stators can be used to pre-swirl the flow, but their effect on the overall performance is less than 1% for angles up to 5 degrees as shown by the prototype experimental results (Hughes 2001).

Bi-directional propeller section

Asymmetrical sections were not to be used as previously explained. The new section was required to have bi-directional characteristics. A new section was developed using X-Foil (Drela 1989).

Over 10,000 sections were automatically created and tested (Ellsmore 2002). A quintic polynomial was used to define the camber line. A leading and trailing edge circle and two cubic splines were used to define the thickness distribution. The section shape was the created by adding the thickness to the camber line (Figure 12). The parameters controlling the camber and thickness were varied over a specified range and a series of bi-directional sections was produced. The sections were then analysed and refinements made to possible candidates.

The developed section has almost the same efficiency (95%) as the standard asymmetrical Kaplan section (Lewis 1988). This improved the efficiency over the old symmetrical section used in the first generation prototype by 3%.

Duct profiles

Different duct profiles were modelled and tested experimentally for the first generation thrusters. The best duct shape (SF2037) was found to be the one that imposed the least increase in velocity at the propeller plane: the section with the flattest inner shape, which agrees with the numerical model.

The duct must enclose the motor, which typically has a length of 100mm and thickness of 40mm. A new duct section shape (MSN64212) was developed with similar inner shape but with sharper ends giving a smoother C_P distribution. This section performs equally well as the SF2037 but will be less prone to separation at the trailing edge, although this is not modelled in the current numerical model.

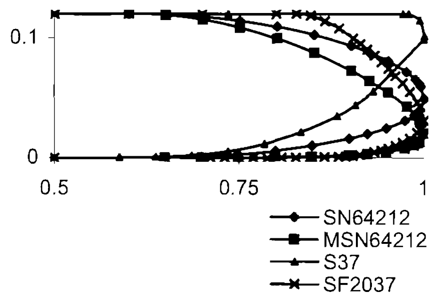


Figure 10 Duct profiles tested

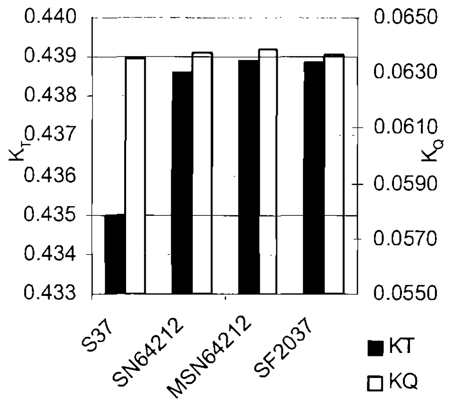


Figure 11 K_T, K_Q for different ducts at J=0.1

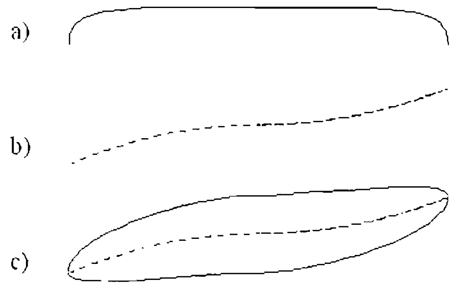


Figure 12 Thickness distribution (a) is added to the camber (b) to generate a symmetrical section (c)

Duct length

Six different duct lengths were modelled at three different advance speeds. As the advance speed increases the optimum duct length increases. For the J=0.6 the optimum duct length is 2.4 times the propeller diameter decreasing to 0.8 times the diameter for J=0.1. The optimum duct length can be selected depending on the application of the thruster, but for a general-purpose case a duct length of 1.8 times the propeller diameter would be best. This gives less than 1% penalty for all the speeds analysed whereas a short duct would have a 3.8% penalty at J=0.6. Since a ROV mostly uses the thruster for position control at low speeds and does not move at high speeds a shorter duct was chosen. Also a compact thruster has practical advantages.

The optimum duct length depends on the thruster size since the thrust of the duct does not scale the same as the propeller thrust (Abdel-Maksoud 2002). The whole problem is Reynolds number dependent and the duct thrust increases with Reynolds number.

A factor not modelled in the current method is the boundary layer effect of the duct on the propeller tip. The longer ducts have a thicker boundary layer at the propeller tip, which increases the loading at the propeller tip (Abdel-Maksoud 2002). However in this case an additional complication is the ring with blade-tip junction. Suitable fairing aids in minimising drag losses in this region.

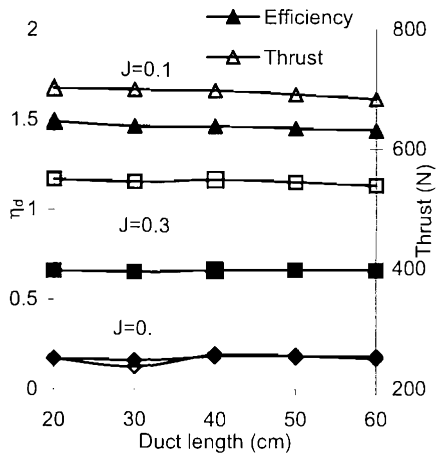


Figure 13 Duct length influence on thrust

Blade area ratio

Since cavitation is not an issue small blade area ratios with highly loaded sections are feasible. Four different BAR of 0.5, 0.6, 0.7, 0.8 and 0.9 where analysed. The pitch was adjusted for each case such that the thrust produced was the same. The chord and pitch distribution along the blade where kept constant.

An increase in pitch is required with a decrease in BAR to keep the thrust constant (Figure 14). In this bollard condition the efficiency can be express better as follows (Lewis 1988):

$$\eta_D = \frac{\left(K_T / \pi\right)^{3/2}}{K_Q} \qquad (3)$$

It can be seen that as the BAR increases there is a reduction of the required pitch to maintain constant thrust. For higher BARs the variation in pitch is very small. The optimum BAR for the analysed advance ratio of 0.1 is 0.77.

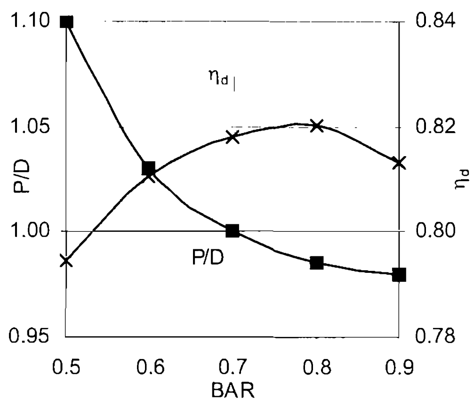


Figure 14 Optimum BAR for J=0.1

COMPLETE THRUSTER

The first generation thruster has been refined and revised. Improvements have been made in its hydrodynamic performance.

The biggest gain in performance was from the new bi-directional section shape which accounts for an improvement in efficiency at bollard pull (η_D) of 5% over the old propeller. Other refinements have made smaller contributions. Some of the improvements are not modelled by the numerical model and are not reflected in the results because of the potential flow assumption. For example, such a gain is the smoother duct shape. Although in the numerical results it appears to have the same performance as the old duct, in reality it will perform better because it is less prone to separation compared to the old shape.

CONCLUSION

Numerical analysis is being increasingly used for design applications. The method outlined in this paper allows results for a particular condition to be acquired in about 15 minutes. However, as demonstrated a significant number of such calculations is required to achieve an overall optimisation. As a result a step-by-step approach has been used to first optimise the components in isolation and then make small changes for the complete problem. Not withstanding the number of assumptions/approximations made in the numerical method, it has been proven as a reliable design guide. The result of this work has been an overall improvement of performance of at least 5%. The final design will be by necessity a compromise between optimum hydrodynamic design and practical/mechanical design issues. As greater experience is gained with each successive generation of integrated electric thruster, it is envisaged that more radical designs will be used and as a result greater performance gains will be possible.

The next step of this ongoing work is to build and test the new thruster. In addition the numerical model will be improved by adapting the wake, which it is believed will improve the numerical predictions.

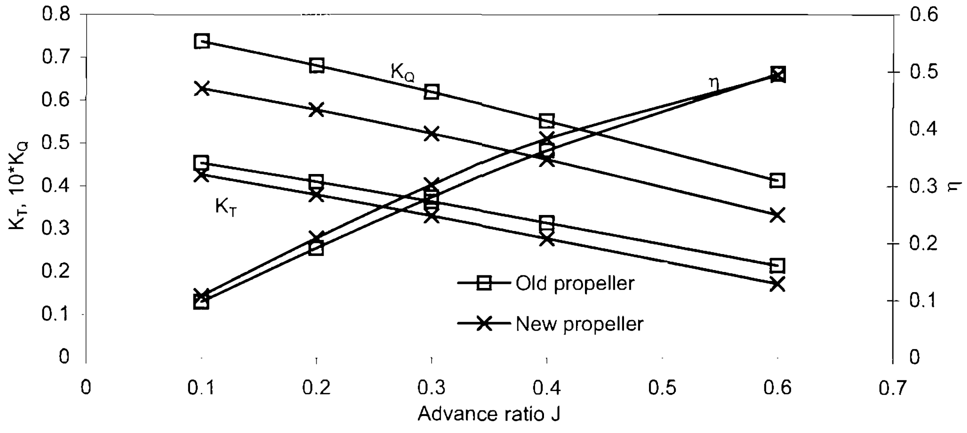


Figure 15 Performance of first and second generation propellers

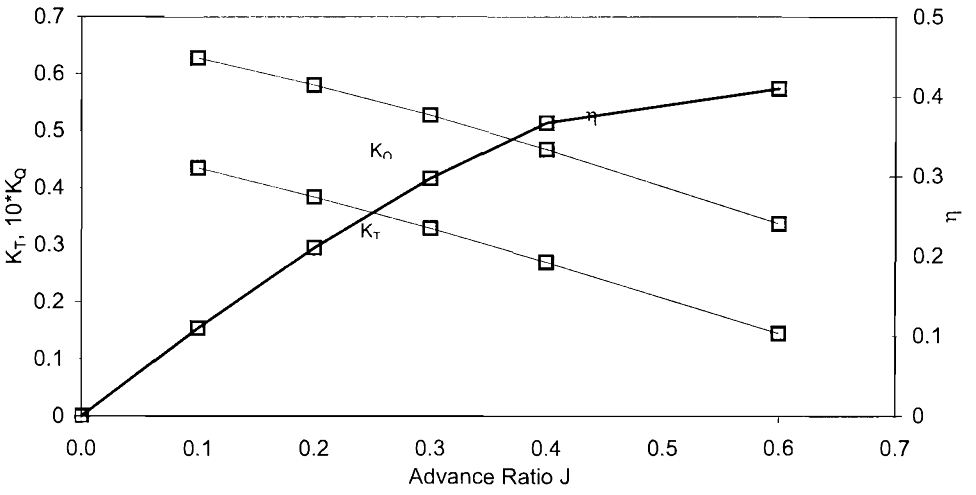


Figure 16 Optimised integrated thruster performance

ABDEL-MAKSOUD M and HEINKE HJ, "Scale effects on ducted propellers." 24th Symposium on Naval Hydrodynamics, Fukuoka, Japan, July 2002

BATTEN W, BRESSLOFF NW and TURNOCK SR, "Transition from vortex to wall driven turbulence production in the Taylor-Couette system with a rotating inner cylinder." International Journal for Numerical Methods in Fluids, Vol. 38 (3), pp. 207-226, 2002

BROWN JB, "Computational approximations for ducted propellers." SNAME Propeller/Shafting Symposium 94, Paper 8.1-18, Virginia Beach USA, Sept 1994

CAPONNETTO M, "The aerodynamic interference between two boats sailing close-hauled." Int. Shipbuild. Progr., 44, no 439: 241-256, 1997

DRELA M, "XFOIL: An Analysis and Design System for Low Reynolds Number Airfoils." Conference on Low Reynolds Number Airfoil Aerodynamics, University of Notre Dame, June 1989

ELLSMORE P, "Optimisation of a bi-directional section." Part III individual project, University of Southampton, 2002

GINDROZ B, HOSHINO T and PYLKKANEN JV editors, "Propeller RANS/Panel Method Workshop Proceedings." 22nd ITTC Propulsion Committee, Grenoble, 1998

HALL CA, "Construction of curvilinear co-ordinate systems and applications to mesh generation." International Journal for Numerical Methods in Engineering, 7:461-477, 1973.

HOSHINO T, "Numerical and experimental analysis of propeller wake by using a surface panel method and a 3-component LDV." 18th Symposium on Naval Hydrodynamics, 1991

HOSHINO T, "Comparative calculations of propeller performance in steady and unsteady flow using a surface panel method." 22nd ITTC Propulsion Committee, Propeller RANS/Panel Method Workshop, Grenoble, France, April 1998

HUGHES AW, TURNOCK SR and ABU-SHARKH SM, "CFD modelling of a novel electromagnetic tip driven thruster for underwater vehicles", ISOPE 2000, Seattle, June 2000.

HUGHES MJ and MASKEW B, "Calculations for the DTMB4119 and DTMB4679 propellers and a highly skewed propeller for the Sein-Maru using the VSAERO/PROFAN and USAERO codes." 22nd ITTC Propulsion Committee, Propeller RANS/Panel Method Workshop, Grenoble, France, April 1998.

HUGHES AW, ABU-SHARKH S and TURNOCK SR, "Design and testing of a novel electromagnetic tip-driven thruster." The Proceedings of the 10th International Offshore and Polar Engineering Conference, Seattle, USA, pp.299-303, June 2000

HUGHES AW, "Investigation of Tip-Driven Thruster and Waterjet Propulsion Systems." PhD thesis, University of Southampton, 2001

JESSUP S, "Experimental data for RANS calculations and comparisons (DTMB4119)." 22nd ITTC Propulsion Committee, Propeller RANS/Panel Method Workshop, Grenoble, France, April 1998

KERWIN JE, "Marine Propellers", Ann. Rev. Fluid Mech., 18:367-403, 1986

LEE TJ, "A potential based method for the analysis of marine propellers in steady flow." PhD thesis, M.I.T. Dept. of Ocean Engineering, Aug 1987

LEWIS EV, "Principles of naval architecture- Volume II: Resistance and propulsion." SNAME, 1988

MCMAHON J, "Characteristics of ducted propellers." SNAME Propeller/ Shafting Symposium 94, Paper 18.1-14, Virginia Beach USA, Sept 1994

MILGRAM J, "Hydrodynamics in advanced sailing design." 21st Symposium on Naval Hydrodynamics, 1997

MORINO L and KUO CC, "Subsonic Potential aerodynamics for complex configurations: A general theory." AIAA Journal, Vol.12, No.2, pp 191-197, Feb 1974

NEWMAN JN, "Distribution of sources and normal dipoles over a quadrilateral panel." Journal of Engineering Mathematics, Vol.20, pp113-126, 1986

PEREIRA F, SALVATORE F, DI FELICE F and ELEFANTE M, "Experimental and numerical investigation of the cavitation pattern on a marine propeller." 24th Symposium on Naval Hydrodynamics, Fukuoka, Japan, July 2002

STRECKWALL H, "Hydrodynamic analysis of three propellers using a surface panel method for steady and unsteady inflow conditions." 22nd ITTC Propulsion Committee, Propeller RANS/Panel Method Workshop, Grenoble, France, April 1998

TAKINACI AC and ATLAR M, " On the importance of boundary layer calculations instead of viscous correction in heavily loaded marine propellers while using a surface panel method." Ocean Engineering, 28:519-536, 2001

TURNOCK SR, MOLLAND AF and WELLICOME JF, "Interaction velocity field method for predicting ship rudder-propeller interaction." SNAME Propeller/Shafting Symposium 94, Paper 18.1-14, Virginia Beach USA, Sept 1994

TURNOCK SR, "Prediction of ship rudder-propeller interaction using parallel computations and wind tunnel measurements." University of Southampton, PhD Thesis, 1993

TURNOCK SR and WRIGHT AM, "Directly coupled fluid structural model of a ship rudder behind a propeller." Marine Structures 13:53-72, Elsevier 2000

TURNOCK SR, "Technical manual and user guide for the surface panel code: Palisupan." University of Southampton, Ship Science Report No.100, Oct 1997