3D Flow Visualisation of a Micro Air Vehicle with Winglets

Sujee Mampitiyarachchi

Supervisor By: Geordie McBain

Department of Aeronautical, Mechanical and Mechatronic Engineering,
University of Sydney

October, 2006

Bachelor's Thesis

Statement of Achievement

- I carried out a literature survey to gain background information and knowledge of past and current work in the field.
- I designed and developed GTSwing library with the guidance of my supervisor.
- All source code for the GTSwing library was written by me.
- Verification of the discretisation schemes was completed by me with assistance from my supervisor.
- All figures, tables and plots in this report were created by me using existing software.
- Conclusions drawn are my own.

This is an accurate representation of the student's contribution

Student …….

Sujee Mampitiyarachchi

Supervisor ……

Geordie McBain

Abstract

The three-dimensional flow about a Micro Air Vehicle with winglets has been investigated using computational fluid dynamics. The wing tip vortex is resultant from the mixing of the high pressure flow under the wing with the low pressure flow over the wing at the wing tips. In low aspect ratio wings, such as those of MAVs, the wing tip vortex effects upto forty percent of the span. Winglets can be used to reduce the impact of the wingtip vortex by pushing it upward and outward of the wing.

The flow solver used is the open-source available Gerris Flow Solver developed by Stephane Popinet. Gerris is a laminar flow solver, solving the incompressible Euler equations. The GTSwing is a library of functions for Matlab and Octave that was developed to enable the three-dimensional representation of wings in the GTS format. The GTSwing libray can generate a three-dimensional representation for a set of airfoil data and planform parameters. The source code for the GTSwing library is listed here, as well as the methodology of its operation.

Verification is always important in computation simulation of physical phenomena, and grid convergence studies for NACA0012 and NACA2412 airfoils were conducted to increase the level of confidence in the discretisation methods used.

In conclusion, the addition of winglets can delay the rollup of the tip vortex. Thus the effect of the sidewash and downwash is reduced. The results show that the effect of the wingtip vortex can be reduced by using a correctly designed winglet.

To my parents Ammi and Thathi

Acknowledgements

Thank you to Geordie McBain for all the guidance, and help provided throughout the course of my thesis.

Thank you to all my friends for the support and counseling provided.

Contents

List of Figures and Tables

Abbreviations

$C_{L_{\max}}$	-	Maximum coefficient of lift
$\alpha_{C_{L_{\max}}}$	-	Angle of attack for maximum lift coefficient
AR	-	Aspect Ratio b^2/S
CFD	-	Computational Fluid Dynamics
GPL	-	GNU Public License, grants any user the right to copy, modify and redistribute programs and source code from developers who have licensed their software under GPL.[*]
GTSwing	-	Matlab/Octave library to generate GTS representation of wings.
MAV	-	Micro Air Vehicle
TIP_ROOT	-	Taper Ratio of Tip Chord over Root Chord
UAV	-	Unmanned Air Vehicle
CFL	-	Courrant-Friedrichs-Lewy stability condition

[*] Full Documentation can be found at http://www.gnu.org/copyleft/gpl.html

Introduction

In the modern aeronautical world, there is significant time and resources spent in the area of unmanned air vehicles. These UAV have a variety of uses in military and civilian operations. A recent development in the field of UAVs has been the Micro Air Vehicle concept. Micro Air Vehicles will provide the ability for an individual to launch and operate an aircraft that will be able to survey the surroundings and collect sensor information. They are intended to operate in close proximity with its targets, deploy rapidly, and interface with their operatives on ground. MAV are restricted to have a maximum dimension of 6 inches, range of 10 kms and endurance of 20-60 minutes.

The dimension constraint of the MAV, coupled with in operation Reynolds range of 5,000 - 100,000 results in an area of aerodynamics that is not optimal for flight. The low-aspect ratio, low Reynolds number aerodynamics has significant obstacles for flight. The lift of low-aspect ratio wings is the result of a linear and non-linear source. The linear lift can be modeled by the circulation around the airfoil. The non-linear source is due to the existent of wingtip vortices that affect much of the span.

There are significant three-dimensional effects as the vortex causes a downwash and sidewash on the wing, effecting the pressure distribution throughout. These tip vortices exist on all flying planforms, however its size and affect is not as significant on conventional planforms with higher AR. Winglets can be used to reduce the effect of the wing tip vortex, by pushing it upwards and outwards from the wing.

The visualization of the three-dimensional flow over a wing/winglet combination representative of a possible MAV planform is the main objective in this thesis. Although in reality all flows are viscid, the viscous effects in a flow about a wing can be reduced to a boundary layer at the surface. This is only applicable if the flow remains attached, once stall occurs the boundary layer becomes detached and influences the flow outside.

Gerris is a computational fluid dynamics solver, which implements the incompressible Euler equations, developed by Stéphane Popinet of New Zealand Institute of Water and Atmospheric Research. Although there has not been thesis research done using this CFD solver in the University Of Sydney School Of Aeronautical, Mechanical and Mechatronic Engineering prior to 2006, there are significant advantages that are gained. It is a Linux application that is distributed under the GNU General Public License, which allows users the ability to alter and revise the source code as they wish.

1.0 Literary Review

The MAV concept is a relatively new development, the term was defined less than a decade ago, and in spite of this there is much research that has been done in this field. There is also much work that was done prior to the existence of the MAV concept, which is also relevant. Vortex flows, low Reynolds number aerodynamics, laminar wall-bounded flows and computational fluid dynamics have all been studied in depth, and however it is the interaction of all these topics that is of interest.

The American Defense Advanced Research Projects Agency (DARPA)[1] defined an MAV to be the largest dimension of 6 inches, and an endurance of 20-60 minutes while carrying a payload of 20 grams. The airspeed is expected to be in the range of 10 – 20 m/s. Along with developing high performance propulsion sources, and weight-saving advance materials, the optimization of the aerodynamics of the lifting surface is critical in achieving these goals.

The flow structures at low Reynolds numbers differ to that of conventional flight Reynolds numbers. These may include separation, reattachment, laminar-turbulent transition and wing tip vortex. The most significant difference is the effect of the tip vortex. The choice of a low enough Reynolds number can avoid the other phenomena and highly weight the tip vortices, as noted in Tang and Zhu[11].

The tip vortex occurs due to the flow interaction of the airflow beneath the wing, the pressure side, combining with the flow over the wing, the suction side. The size and shape of the vortex depends greatly on the wing planform. In aircraft with wings of conventional aspect-ratios of 6-10, the vortex has little or no effect over the majority of the wing hence is often ignored. The tip vortex modifies the local flow by inducing an additional vertical velocity that reduces the local angle of attack.

On low aspect ratio wings, the tip vortex has a significant effect on the lift distribution of the whole wing. Tip vortex reduces the local effective angle of attack, hence reducing the lift. Viieru et al.[2] found that the tip vortex effect up to 30% of the wing is affected. Mönttinen[3] in his dissertation states that up to 50% of the wing is affected. Thus the flow over a MAV wing cannot be approximated as constant 2-dimensional flow irrespective of the spanwise location.

There has been some research which suggests that the tip vortex energizes the flow on the upper wing surface in the local region of the wing tip. Sathaye[14] found that a separation bubble formed on the upper surface of AR = 1 wing, however this did not extend spanwise to the local area of the wing tip. Torres and Meuller[13] proposed this to explain a noted transition of $C_{L_{\max}}$ and $\alpha_{C_{L_{\max}}}$ in the AR range between 1.25 and 1.5. For wings below 1.25, the tip vortex is able to delay the onset of separation. However, as the aspect ratio increases the effect decreases. This is considered as the non-linear lifting source, which is results in a high value of stall angle of attack, as concluded by Mueller and DeLaurier[17] .

The low-aspect ratio is a required due to the dimensional constraint and the need to achieve enough lift. As wing span cannot be increase, the chord of the wing is the only variable in determining surface area. Current MAVs have AR in the range of 0.50 – 2.00. Reducing the effect of the wingtip vortex is greatly important in achieving a successful design.

The choice of the airfoil of the wing to be modeled was difficult, due to conflicting requirements in MAV design. Pelletier and Muellller[6] used airfoils with thickness-to-chord ratio of 1.93% in their investigation into the aerodynamics of MAV, citing research done by Selig et al[7]. However these airfoils are often impractical, as the wing needs to support the power plant, batteries, receiver and the payload of the MAV. The Bidule concept, developed by Spoerry and Wong[8], used the significantly thicker NACA 4418 section.

The wing planform has significant influence on the low Reynolds number aerodynamics, including centre of lift, maximum lift coefficient and maximum lift curve slope. Torres and Meuller[13] published a thorough study of characteristics of low aspect ratio wings. They concluded that for AR below 1.0 and high angles of attack, Zimmerman planforms were more aerodynamically efficient. Elliptical planforms performed better at higher AR and low angles of attack. The Zimmerman[16] planforms are produced by joining 2 half-ellipses at either the ¼ chord or ¾ chord locations.

Winglets are the major tool in reducing the effect of the tip vortex. Winglets have been implemented on a variety of aircraft, usually an afterthought to reduce fuel consumption in cruise or reduce aircraft wake effects. On a MAV, the winglet becomes an integral part of the lifting wing, and must be designed so. The mechanism of how the winglet affects the wingtip vortex is simple. The vortex is pushed upwards and out by a vertical velocity component.

There are many types of winglets, spiroid, flat-plate or blended, and the size, shape and orientation of the winglet determines how it effects the tip vortex. Mönttinen[3] found flat-plate winglets increases the pressure drag, but leads to an overall reduction in drag. Viieru et al[2] observed that the flat end-plate increases lift without a significant increase in drag. Mönttinen[3] also found that winglets with chord covering only part of the wingtip to be more effective.

Gerris[5] is a computational fluid dynamics solver, which implements the incompressible Euler equations, developed by Stéphane Popinet of New Zealand Institute of Water and Atmospheric Research. It is a Linux application that is distributed under the GNU General Public License. It contains adaptive mesh refinement which is ideal for the nature of the flow to be studied. A wingtip vortex is a translationary vortex; hence a mesh that is able to evolve with the flow would be more efficient and may be more accurate.

Gerris is a laminar flow solver, it does not implement a turbulence model. Mönttinen[3] observed the pressure field around the wing differed very little in turbulent and laminar computations. Carmichael[15] noted in his low Reynolds number survey that in the Reynolds numbers between 1,000 and 10,000 the flow is strongly laminar and it is difficult to produce a turbulent boundary layer. Micro Air Vehicles operate in the range of Reynolds numbers of 10,000 – 100,000.

Initially it was planned to use Tang and Zhu[11] research to verify the phenomena observed in this visualization study, however it later became obvious that could not happen. Tang and Zhu used high angles of attack to highlight the wingtip vortex, and due to the in viscid nature of this study, significant flow structures would not be resolved at this high angle of attack. Subsequently Mönttinen[3] findings were used extensively to verify results obtained.

2.0 Problem Definition and Methodology

At the tips of the wings, the flow over the upper surface and the lower surface interact with the flow beside the wing forming the wingtip vortex. The flow over the wing is low pressure and the flow beneath is high pressure flow, the flow beside the wings is of a pressure between. Due to this pressure gradient the flow over the wing is pushed inward and the flow beneath is pushed outwards as is illustrated below in Figure 2.1.

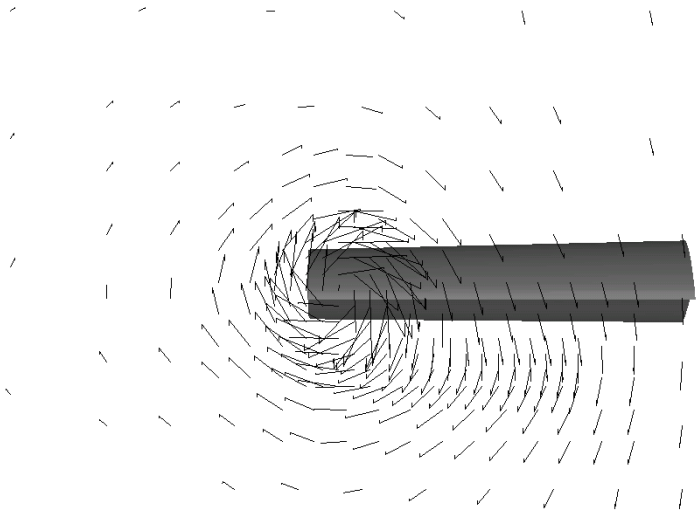


Figure 2.1 Mixing of the airflow at the wingtip

This flow structure can only be modeled in a three-dimensional visualisation due to the sidewash and downwash. The tip vortex effects the pressure distribution along the wing, and subsequently the lift generated. The significance of effect of the wing tip vortex depends largely on the aspect ratio of the wing. On wings of high AR, the wingtip vortex is often largely neglected due to its effect localized to the outboard sections of the wing. However on low AR wings, such as those of micro air vehicles the wing tip vortex affects up to 40 % of the span. Winglets can reduce the effect of the wingtip vortex by moving the vortex upward and outward of the wing. Two winglet models will be tested to determine their effect on the wingtip vortex.

A model of the Micro Air Vehicle wing with winglet attachment needs to be developed and represented as required by Gerris. This model is a complex shape of non-planar surfaces and hence generating the GTS representation is not straightforward.

Mesh generation is the next step in the analysis process. Validation of the discretisation schemes is very important when simulating physical phenomena; hence a grid convergence study must be undertaken. Ideally experimental or analytical results are to be used in the validation.

Verification of the results using published literature will need to be conducted subsequently.

2.1 Gerris Flow Solver

The Gerris Flow Solver is open-source software developed by Stephane Popinet of the New Zealand National Institute of Water and Atmospheric Research. It is a laminar flow solver, implementing the incompressible Euler equations. Gerris is still in active development, and this study used the Gerris-0.9.2 and Gerris-0.9.3 releases, it is primarily operated in a Linux environment. GfsView is a post-processing tool that is packaged with Gerris, and is used to visualize the resultant output from the solver.

In Gerris the domain is discretised into square or cubic finite volumes that are stored respectively in quadtrees and octrees. Each finite volume or cell can be the parent of up to four child cells, eight in three-dimensional. A leaf cell is one with no children and the root cell is the cell with no parent. This use of a hierarchy tree to store cells is straightforward and allows efficient access to neighbouring cells and traversal of cells at a certain level or depth in the tree. As stated by Popinet[5], the following restrictions apply on the cell level:

- The levels of neighbouring cells cannot differ by more than two,

- All the cells directly neighbouring a cell cut by the solid boundary must be at the same level.

Adaptive mesh refinement is an important feature of Gerris, allowing the grid to develop with the evolution of the flow structures. It reduces complexity of the mesh scheme and increases efficiency, as cells are refined only when needed.

Refinement and coarsening of the grid by is straightforward and efficient due to the quadtree representation. All cells which have a gradient greater than the specified are split. Cells which have a gradient less than the specified are removed and their parent is made a leaf cell. This calculation is conducted only once each timestep, as it is assumed that the flow is evolving slowly.

The advection term is evaluated using the Godunov procedure, with the classical Courant-Friedrichs-Lewy stability condition. This is to prevent overflow of a cell in any given timestep.

Gerris is a scripted program, requiring a GFS script to run a simulation. This GFS script defines the layout of the domain, initial conditions, boundary conditions, the solid boundaries, parameters of the discretisation schemes, when to output, what to output and where to output. Examples of Gerris script files are listed in Appendix C.

The boundary conditions used was an inlet condition on the upstream XY face, an outlet on the downstream XY-plane, and symmetrical conditions on the remaining 4 boundary planes. The no-slip condition is applied to the solid boundary. This set of boundary conditions is used for the two-dimensional and three-dimensional modeling.

An in viscid visualization of the flow was attempted; however it failed as Gerris has the limitation of currently being unable to solve the diffusion problem when there are mixed boundary cells. This functionality is scheduled to be completed in the next Gerris release.

A full-span model would not contain mixed boundary cells; however the computation would require a high level of time and resources that exceed the available.

2.1.1 Problems with Solid Force

When conducting a grid convergence study is became necessary to determine the pressure force acting on the solid. Gerris contains the output command GfsSolidForce, which will output the x, y and z components and the moment components acting on the embedded solid. The y-pressure coefficient values obtained using the GfsSolidForce output command was vastly different from what was expected. Despite this the pressure field, as viewed in GfsView, is comparable to what is expected. Thus, another method of obtaining the pressure acting on the solid was developed.

In the generation of the solid geometry using the GTSwing library, the section profile at the 50% span location was saved into a file. This file contains a series of points with x, y, z coordinates defining the section perimeter. Using the GfsOutputLocation command, all the values of the permanent variables at the locations specified in the file are obtained. The pressure was plot in a pressure distribution plot and compared to the pressure field observed in GfsView. There again was a discrepancy, with a symmetric pressure field being observed and an asymmetric pressure distribution. The square of the velocity was observed to be symmetrical and hence this is trusted method of observing the pressure on the solid.

2.1.2 Resource Management

During the Gerris familiarization process, the amount of resources in terms of time and computer memory used to compute a solution was important. There was numerous times where a simulation was terminated without warning. After reviewing the output logs, the cause of this termination was not determined. A possibility was a lack of memory.

To observe the memory statistics during a run of a simulation, the GfsEventScript command was used. This allows for shell commands to be scheduled at certain points in the run.

```
GfsEventScript {istep = 10} {
      echo –n $GfsTime >> memloading.txt
      free | gawk '/Swap/ {printf " %d\n, $4}' >> memloading.txt
      }
```

An observation on the level of memory allocated to the process and memory free showed that the simulations were using 100 % of the RAM, and using additional memory in the Linux-swap-file. Figure 2.2 below displays the swap memory free during a Gerris computation. There is adaptive mesh refinement; hence the number of cells in the domain is changing, due to the flow evolution the cells are increasing.

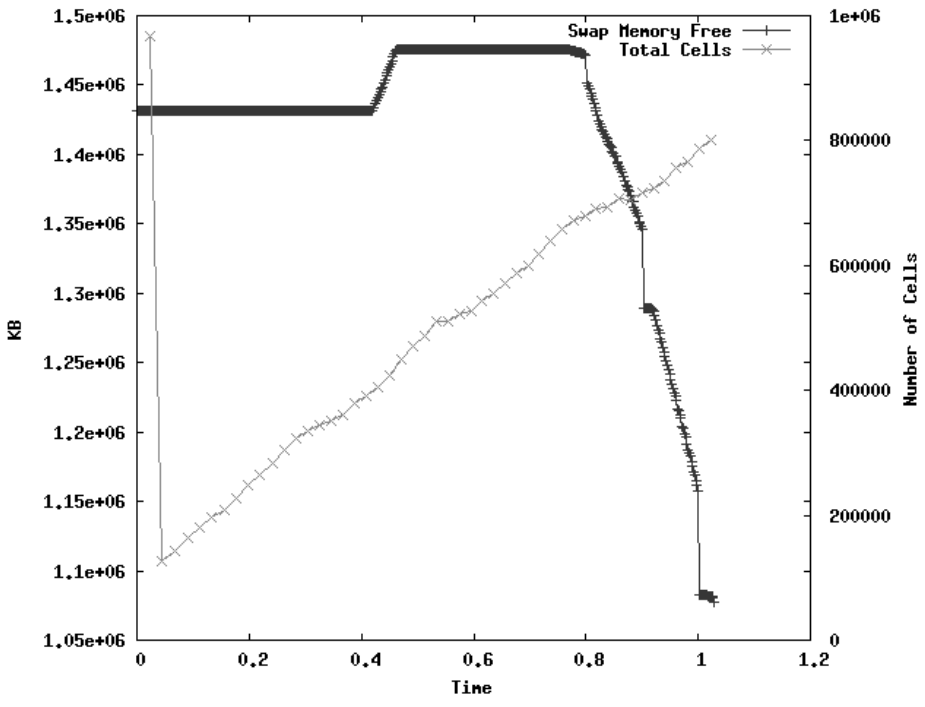


Figure 2.2 Memory Usage

This resulted in lengthy computations, taking days for a two-dimensional visualization and drastically longer for three-dimensional. The MAV itself is symmetric, and at low

angles of attack the flow is symmetric. Taking advantage of this, it was decided to model only the semispan and use a symmetric condition. This restricted the solid orientation to low angles of attack. It was also decided that it would be sensible to limit the maximum number of cells in the domain, as it would result in faster computations. A maximum cell count of 250,000 was applied to restrict the memory needed to the amount of RAM available on the local machine and prevent a high consumption of virtual memory..

2.2 Other Computational Tools Used

GfsView is a post-processing tool used to visualize the simulation files that are output from Gerris. This was used extensively throughout in conjunction with Gerris. It contains functionality that allows users to write a GFV script which allows GfsView to run a specified set of commands.

Matlab and its open-source equivalent Octave were used extensively in the development of the GTSwing library and for various preprocessing and postprocessing. Gnuplot was used to generate the plots used throughout this thesis.

Blender and Geomview are open-source three-dimensional rendering application and were used to inspect the solid geometry, and obtain illustrations.

Admesh in addition to the conversion utilities packaged with the GTS library were used to inspect, translate, rotate and scale the triangulated surface files. Admesh is available open-source.

3.0 Solid Geometry

An accurate representation of the solid to be modeled must be determined when conducting flow visualization. In Gerris, objects in the fluid domain must be represented by a GTS file. GTS is the abbreviation for the GNU Triangulated Surface Library. It is not a widely used format amongst the CAD programs. A GTS representation of a surface has the following format:

nop nol not GtsSurface GtsFace GtsEdge GtsVertex
x y z Cartesian coordinates of each vertex
Starting and ending vertex of each edge
3 edges forming each triangle

Where *nop*, *nol*, and *not* are the number of points, edges and triangles respectively. This surface can now be read by Gerris. The breakdown of a solid into its GTS representation is difficult when dealing with complex shapes. As the GTS representation consists of only triangles, this breakdown of the solid is referred to as the triangulation of the solid.

The Gerris package does not contain any utilities for creating GTS files of solids, more complex than spheres or cubes; however it does contain tools for converting an STL triangulated mesh into a GTS representation. The STL format is compatible with a wide range of CAD programs and 3D rendering software. The following two options were available to generate the Solid Geometry:

1. Model the solid in a CAD program, such as SolidWorks, export as STL, and convert into GTS format.

2. Develop a set of algorithms to generate a GTS representation of a wing, defined by airfoil coordinate data and planform parameters.

The second option was chosen, due to it being a faster process, easily repeatable and customizable. The Matlab and the compatible open-source equivalent Octave was chosen

as the medium of implementation due vectorization, and the vast libraries of mathematical functions already existent.

The end result of this process is the GTSwing library for Matlab/Octave, the source code of which is listed in Appendix B. The solid geometry generation can be categorised into three configurations:

- Rectangular Wing.
- Elliptical planform
- Winglets attached at wingtip

3.1 Airfoil Data

The airfoil to be used is specified by a matrix of 2 columns [x, y] containing coordinates that make up the airfoil profile. These matrix cannot contain duplicities, must be ordered from the trailing over the wing to the leading edge and under the wing. Airfoil data listed in Abbott and Von Doenhoff[19] and NASG Airfoil Database[21] have the same ordering.

The NACA 4 series camber equations and thickness equations have been implemented in *naca4series* and hence, NACA 4 series airfoils can be generated that conform to the requirements. Problems can occur in the triangulation when there is a high number of points used to defined in the airfoil data set. The *neaten* function was developed to redefine the airfoil in a specified number of datapoints. The spline of the original airfoil dataset is used to obtain the new set, thus there is no loss in accuracy.

3.2 Rectangular/ Trapezoidal Wing

It is important to note that only the surfaces of the solid in contact with the fluid need to be represented. This maybe termed the solid boundary. The solid boundary of a wing

totally immersed in the fluid can be thought of as 2 section profiles and the joining surface.

The joining surface is defined by the airfoil coordinates and its spanwise location. If the spanwise location of one section profile is taken as zero, then the other is equal to the span of the wing. Also the vertices defined by the airfoil section data and the spanwise location of the section, are all the vertices needed for the GTS representation. Hence for a constant section wing, the number of vertices is equal to twice the number of points defining the airfoil section. For a rectangular planform the joining surface is also rectangular.

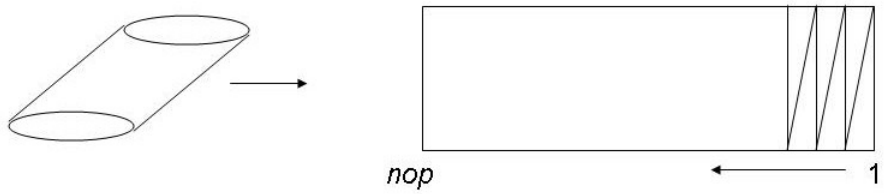


Figure 3.1 Triangulation of joining surface

The algorithm for creating the edges is as follows:

for 1 to number of points,
Join point(i) to point(i+1)
Join point(i) on other profile to point(i+1) on other profile
Join point(i) to point(i) on other profile
Join point(i+1) to point(i) on other profile
end

These lines can be thought of as joining the current point to the adjacent, orthogonal and diagonal points. Using vectorization in Matlab or Octave, the implementation is as follows:

```
% Using a points array of length 2*NoP which stores all the vertices.

i=1:NoP;
lines = [i,i+1;
```

```
NoP+i, NoP+i+1;
i, NoP+i;
i+1, NoP+i];
```

Now that all the lines have been joined, then the algorithm for creating the triangles can be determined. It is helpful to look at the layout of the lines array at this time:

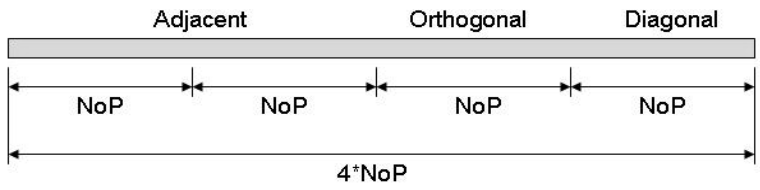


Figure 3.2 Layout of the lines array

for i= 1 to number of points-1,
*Triangle is formed by line(i), line(3*NoP+i) and line(2*NoP+i)*
end

The order of the lines forming the triangles is important, as this determines the direction of the normal. The normal of all the triangles forming the GTS surface must be consistent, either all inwards facing or all outwards facing. Triangulation of the joining surface between two section profiles is implemented in *gtsExtrude* .

Triangulation of the section profile is similar to the triangulation process undertaken above. This is implemented in *gts2D*, which takes the section profile as the input. Care needs to be taken with the lines forming the leading most triangle and trailing most triangle.

```
t=[1:NoP/2-2]';

tris=[t, NoP+t, 1.5*NoP-2+t;          % Lower Triangles [adj,dia,ortho]
    NoP/2-1, NoP/2, 2*NoP-3;          % LeftOver Lead Triangle
     NoP/2+t, 1.5*NoP-1-t, 2*NoP-2-t; % Upper Triangles[adj,dia,ortho]
     NoP-1, NoP,1.5*NoP-1];           % LeftOver Trailing Triangle
else
```

Now the two section profiles and the joining surface forming the complete solid boundary of the wing has been triangulated. To join the 3 surfaces together to form the solid boundary, the *gtsmerge* function was developed.

The *gtsmerge* function takes two GTS surfaces, a and b, as input and returns GTS surface c, which is the result of joining the input surfaces. The two surfaces must have a well-defined join; the intersection of faces is not handled. The point array of surface a is copied into the output points array, then the points array of surface b is appended, noting the new indexes of the vertices in a points map.

$points_A\ (i) = points_C\ (i)$
$points_B(i) = points_C(points\_map(i))$

The lines array of surface A can be read directly into the lines array of surface C as the indices of the points from surface A have not been changed in the points array of C. The relationship of the points arrays are illustrated above. However the lines array from surface B has to be redirected to the new indices.

$lines_B\ (i) = [\ 1,\ 2\ ]$
$lines_C\ (k) = [\ points\_map(1),\ \ points\_map(2)\ ]$

It is important to note that the index of the particular line itself changes; hence this needs to be stored in a lines map. This is similar to the points map. The triangles array of surface A can be read in directly, as the indices have been maintained. The triangles array of surface B has to be redirected as above.

$tris_B\ (i) = [\ 4\ 5\ 7]$
$tris_C\ (k) = [\ lines\_map(4),\ lines\_map(5),\ lines\_map(6)\ ]$

The merging is now complete. Earlier it was mentioned that there needed to be a well-defined join, this means that there needs to be a plane of vertices common to both surfaces. This process in actuality, maps one GTS surface in to the frame of reference of another, the merging is by product if there are vertices that are common to both.

3.3 Elliptical Planform

Elliptical, and any other complex non-rectangular, planforms can be approximated by discretising the planform into rectangular/ trapezoidal segments. Each segment has a constant sweep, twist, dihedral and taper ratio and can be triangulated using the method outline in the previous section. The taper ratio, sweep, x and z offsets of these segments need to be calculated so that the elliptical shape is still maintained, as it done in the *epwing* function and illustrated below. It is significant to note that the segments do not have a constant span, rather the x locations are sinusoidally distributed to the wingtips. This distribution is more efficient than a uniform spanwise distribution as sections that have more curvature have smaller spans hence better defined.

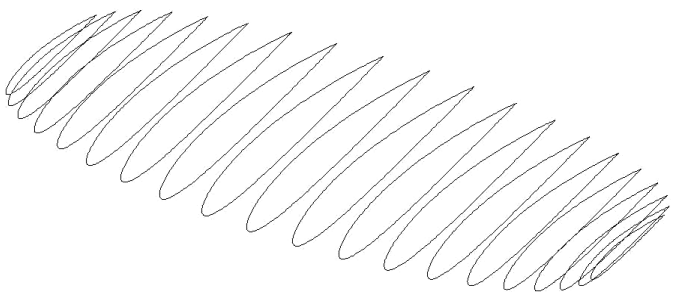


Figure 3.3 Elliptical planform is discretised.

Then, the above procedure can be used to mesh these individual segments and then joined to produce the elliptical wing. The joining of the individual segments was initially done in Blender; however this was a slow process as it required converting to and from STL format. To replace this process, the *gtsmergeall* and *gtsmergefile* functions were developed. These use same *gtsmerge* function describe previously. As mentioned

previously they do not strictly merge surfaces; rather these functions allow one GTS surface to be mapped into the same frame of reference as another.

The wing tip sections are of special significance, as they may contain profiles with zero chord length. The function *gtsEx2Vert* was developed to create a surface from a finite chord to a chord of zero length. This surface can be created simply by replacing the chord of zero length by a tip vertex, and joining all the points of the finite chord to it.

This ability to triangulate elliptical and other non-quadrilateral wing planforms was initially thought to be needed; however this functionality was subsequently not used.

3.4 Addition of Winglets

Any of the wings generated using the methods described previously, can have winglets attached, given that there is a non-zero wingtip chord. This aspect of the solid geometry proved to be the most difficult. Initially, the winglet was modeled as another segment with a unusual section profile. Problems arose with maintaining a smooth extrusion from an airfoil section to the winglet section profile.

The final method that is used treats the winglet as a section of the span in the vicinity of the wingtip with a dihedral of 90°. This completely removed the problems that were encountered using the previous method.

The transformation steps involved in this method are listed below:

1. Scale and offset the wingtip section according to the specified parameters
2. Rotate this new section by 90° or -90°, depending on the port or starboard wing tip.
3. Apply a bend radius; otherwise there will be problems in the extrusion.

4. Triangulate, using gtsExtrude, the surface formed by the wing tip section and the winglet base section.
5. Taper and offset the winglet base section to form the winglet tip section.
6. Triangulate, using gtsExtrude the surface formed by the winglet base section and the winglet tip section
7. Triangulate the winglet tip section, using gts2D.
8. Merge all three in a single GTS representation.

The following figure, illustrates the wingtip section, winglet base section and winglet tip section, as well as the completed triangulated model of a winglet.

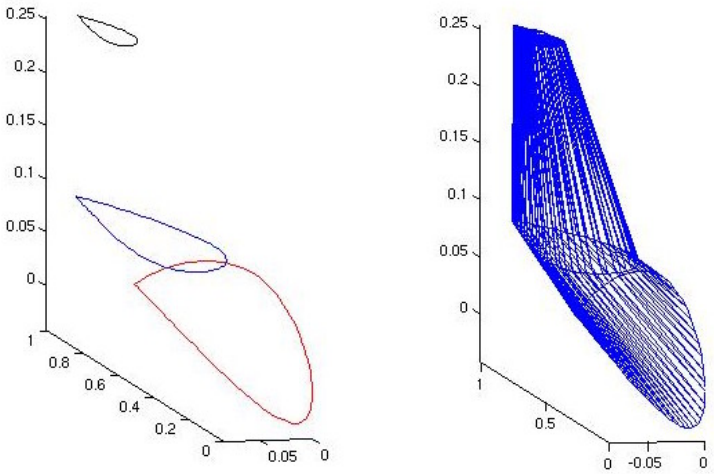


Figure 3.4 Winglet triangulation

3.5 GTSwing-0.0.11 Timing Summary

One of the reasons for developing the GTSwing library rather than using a CAD model to represent the MAV was that it would be a faster process. A timing test was conducted to determine if this was actually was true.

Tests Conducted on 13/9/06
Testing Hardware

Intel M 1.7 ghz Acer TravelMate 4150
1 mb cache, 500 mb RAM
Running Linux 2.6.16.5-kanotix-2
Matlab R14 for
Linux
Airfoil Data obtained from naca4series

Definitions

NoP	number of datapoints in the airfoil data
NoS	number of segments wing is discretised into
Section	1 for closed, 0 for open tip section
etime	elapsed time
epwing	elliptical planform wing

Table 3.1 Timing Results for *epwing*

Planform	NoP	NoS	Section	etime (s)
epwing	102	20	1	59.06
epwing	102	20	0	55.45
epwing	102	10	0	14.12
epwing	102	10	1	15.83
epwing	102	40	1	225.26
epwing	102	40	0	218.72
epwing	202	20	0	199.92
epwing	402	20	0	762.51

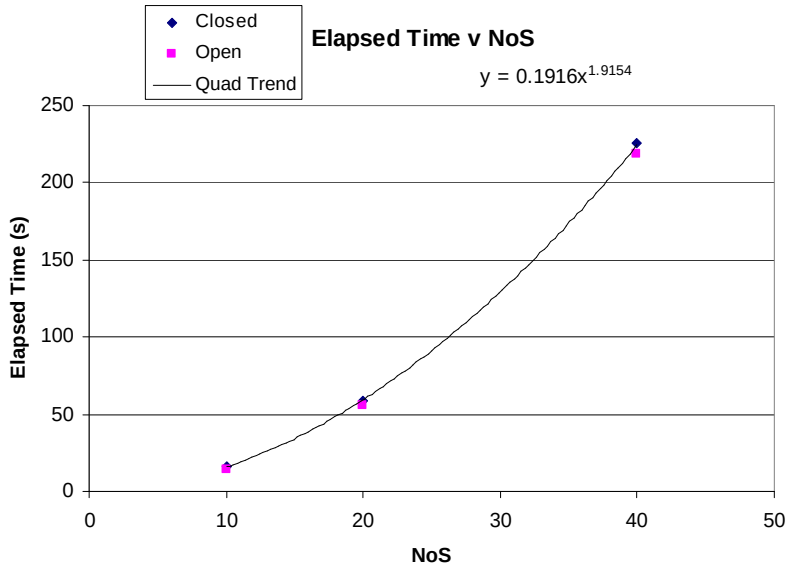


Figure 3.5 Elapsed time variation with number of segments

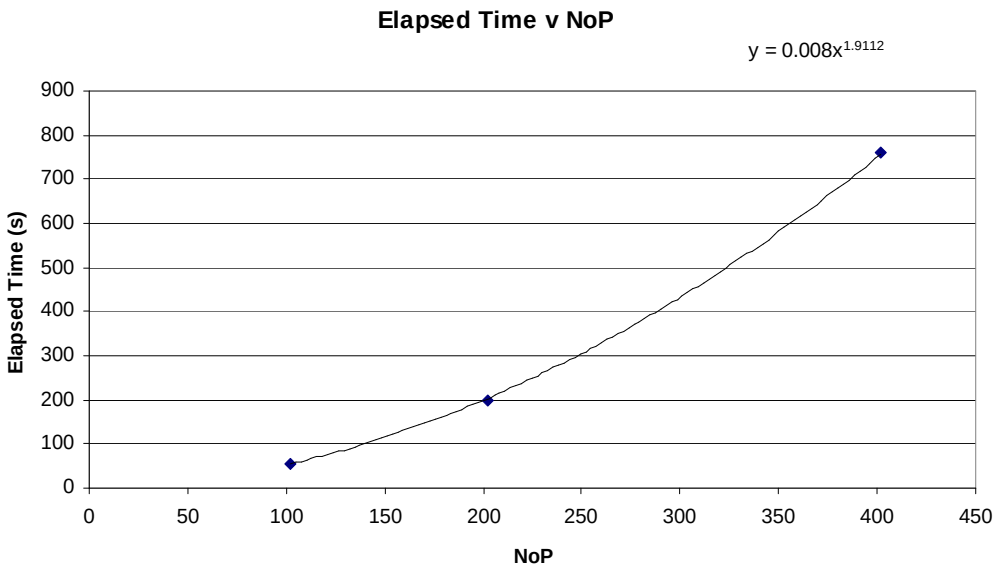


Figure 3.6 Elapsed time variations with number of points.

The tests were conducted on the most complex planform, hence rectangular/ trapezoidal planforms are much quicker to generate. The elapsed time is more sensitive to the number of segments in the planform discretisation than the number of points used to define the airfoil profile. Using the GTSwing library to generate GTS representations of three-dimensional wings is more efficient than using CAD software.

3.6 Models Used

The models used in the study are discussed and defined in this section.

3.6.1 Two-Dimensional Grid Convergence

Gerris requires all solids in the fluid domain to defined in the GTS format, this is a three-dimensional representation. All solids must be three-dimensional objects. The wings used

in the two-dimensional convergence studies were rectangular wing of aspect-ratio 6 and zero leading sweep and taper-ratio of one.

3.6.2 Three-Dimensional Study

There was no specification on the solid geometry to be analysed, other than it to be representative of a MAV. The baseline wing used in three-dimensional simulations is the same as Mönttinen[3] used a tapered and swept wing with the Eppler 212 airfoil section. The Eppler 212 airfoil data can be found in Eppler[20]. This was chosen as to enable comparison to the Mönttinen[3] study. The Figure 3.7 below illustrates the low aspect ratio planform and cambered airfoil.

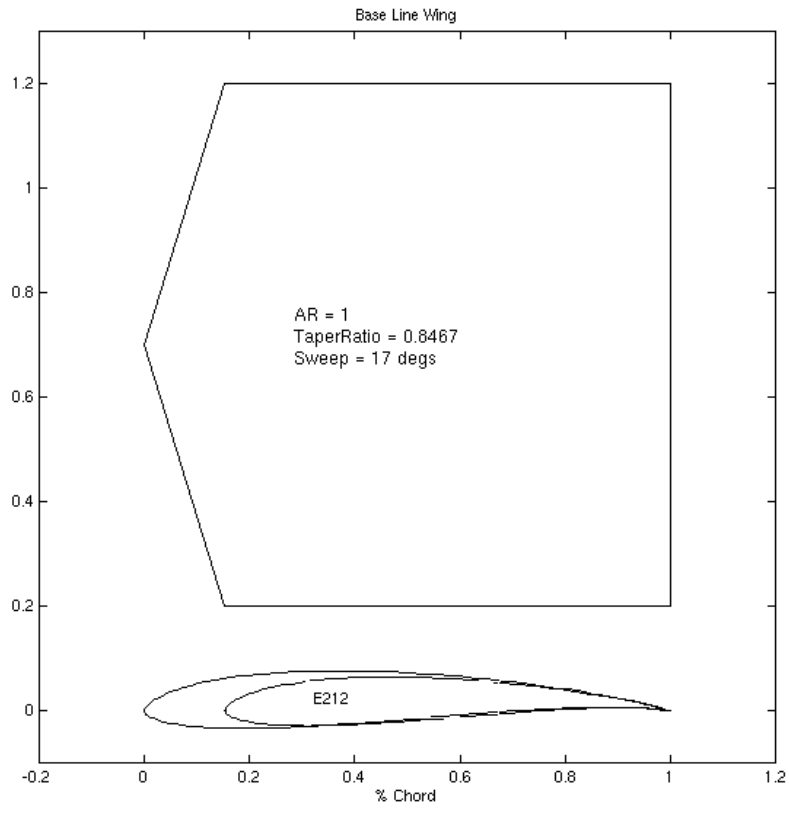


Figure 3.7 Planform and airfoil shape of base wing.

Attached to the base wing were the following winglet models, developed using similar parameters to those used by Mönttinen[3]. These attach to the tips of the base wing. The two winglet models used are described below in Table 3.2. The significant differences being the span or height of the winglet and the taper ratio of the winglet tip to the base. The choice of airfoil section was based purely on the efficiency of its attachment to the base wing.

Table 3.2 Winglet Models

Parameters	**Winglet A**	**Winglet B**
Label	wA	wB
airfoil	Eppler 212	Eppler 212
Span (height)	0.169	0.12
Sweep Leading Edge.	67.5°	65°
Taper Ratio	0.4	0.66
x_offset	0.32	0.25

4.0 Meshing the Domain

Once an accurate representation of the solid geometry has been achieved, the next step is the meshing of the fluid domain. A well-balanced mesh is vital in achieving relevant results; a too fine a mesh will result in high consumption of time and memory resources and a too coarse mesh will not resolve the flow structures correctly. It is favorable to have a high level of refinement in the local region around the airfoil, and a coarse mesh in the far field, this allows for the flow to be resolved and an efficient simulation.

The fluid domain a rectangular prism with dimensions 5 × 5 × 15 chords. The left face is a velocity inlet, and the right end is an outflow. Symmetry conditions are imposed on the other faces of the domain. For a semispan model, the solid is located -2.5 chords from the velocity inlet, midway in y and on the back wall.

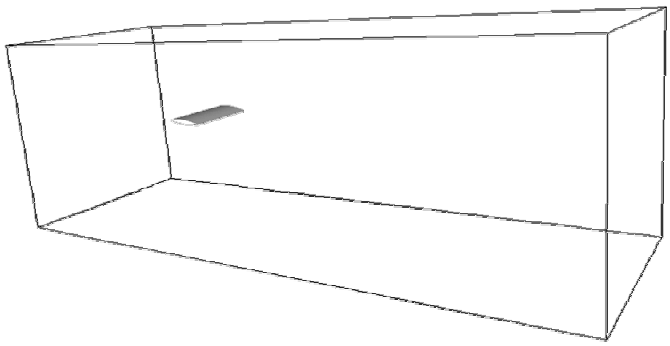


Figure 4.1 Domain Boundary

There are a number of commands in Gerris to define refinement levels, and often combinations of these commands are used to generate a balanced mesh:

- GfsRefineSolid- for cells cut by the solid boundary
- GfsRefine – for all cells in the domain.

The refinement level of cells adjacent to the wing is defined using the GfsRefineSolid command. The actual level of refinement can vary along the solid boundary, limited by the constraints on neighbouring cells as mentioned in section 2.1. For example in the following code extract, the level defined by GfsRefineSolid is depended on the x-location of the cell. The chord of the wing is from 0 to 0.2; hence the leading edge and trailing edge have higher levels of refinement than the rest of the solid.

```
GfsRefineSolid {
return x< 0.02 ? 12 :  x > 0.18 ? 10 : 9
}
```

Once the initial mesh has been set, adaptation may be included by using GfsAdpaptVorticity or GfsAdaptGradient, which will split or merge cells depending on the gradient limit specified. Using grid adaptation will reduce the number of elements in the domain; hence reduce computational time and resources. Below is an observation of the average number of cells in the domain at every tenth iteration of a 2D solution of flow over a $NACA65_{a=0.5}212$ airfoil.

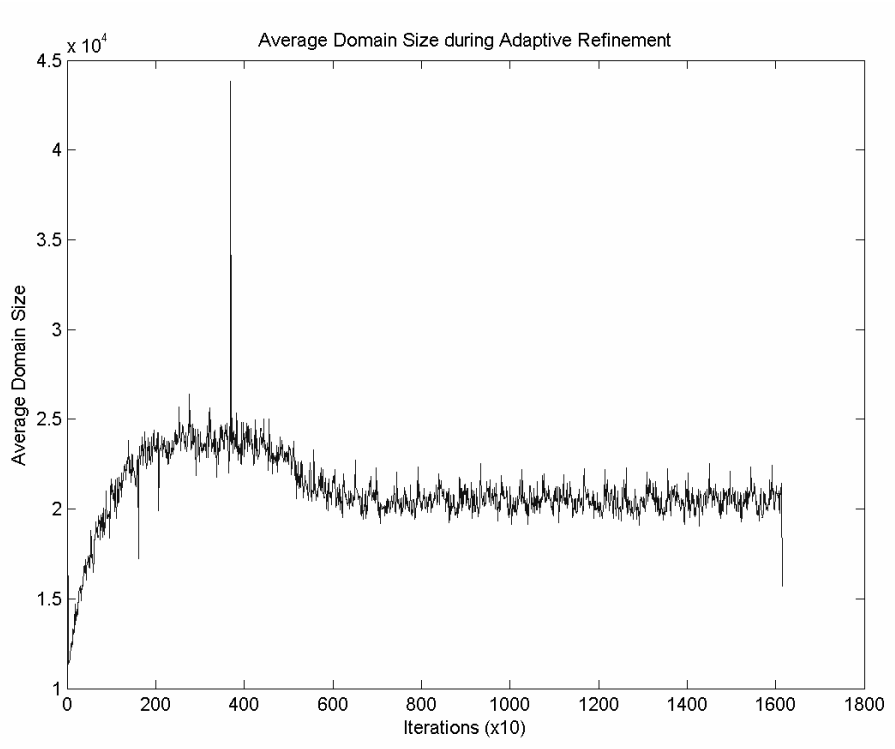


Figure 4.2 Adaptive Grid Refinement.

The following adaptation was used:

```
GfsAdaptGradient { istep = 1 } { minlevel = 4 maxlevel = 10 cmax = 1e-2
} P
```

If the flow has reached a steady state condition, then the fluid domain grid would not change with time. As the above figure displays oscillatory behaviour after 700 iterations, the flow has reached steady state or pseudo-steady state conditions.

4.1 Problems at Leading Edge

Gerris uses square, cuboid in three-dimensional, finite volumes to discretise the domain; solid boundaries that are not aligned with these will not be resolved exactly. Solid boundaries with significant curvature require square cells of a high refinement. The leading edge of an airfoil is a region of high curvature; it is also the region where the

majority of the lift is generated. Thus, for the leading edge to be model accurately there needs to be a high level of refinement in the cells cut by the solid boundary.

An inspection of the vorticity field in the vicinity of the leading edge displays unexpected vorticity. Increasing the local grid refinement reduces the scale of this superfluous vorticity.

4.2 Kutta-Joukouwsky Condition

The flow about an airfoil can be considered as a uniform translatory flow superimposed by a circulatory flow. The Kutta-Joukowsky condition states that the flow leaving the trailing edge must be smooth, as the circulation adjusts itself till it is so. The streamlined trailing edge is difficult to model accurately in Gerris due to it sharpness. As mentioned in the previous section, solid boundaries of high curvature will require a high level of refinement.

Investigation of the solid itself revealed a truncation error in the airfoil dataset used to generate the wing. The airfoil data which were obtained from Abbott and Von Doenhoff[19] did not contain a trailing edge of zero thickness. The truncated trailing edge resulted in the airfoil not comforming to the Kutta-Joukowsky condition. Using the analytical NACA camber line function and NACA thickness function, a streamlined airfoil conforming to the zero-thickness trailing edge criterion can be generated.

$$y_t = \frac{t}{0.2}\left(0.2969\sqrt{x} - 0.126x - 0.35160x^2 + 0.2843x^3 - 0.1015x^4\right)$$

$$y_c = \frac{m}{p^2}\left(2px - x^2\right) \qquad \text{for x < p}$$

$$y_c = \frac{m}{(1-p)^2}\left((1-2p) + 2px - x^2\right) \qquad \text{for x > p}$$

In the above equations p is the position of max camber, t is the maximum thickness and m is the maximum camber. The trailing edge is defined to be the root of the thickness equation, and the x stations are distributed subsequently, this is implemented in the function *naca4series* and *nacathickness*. Figure 4.1 illustrates streamlined trailing edge and the consistent profile shape for the remainder of the airfoil.

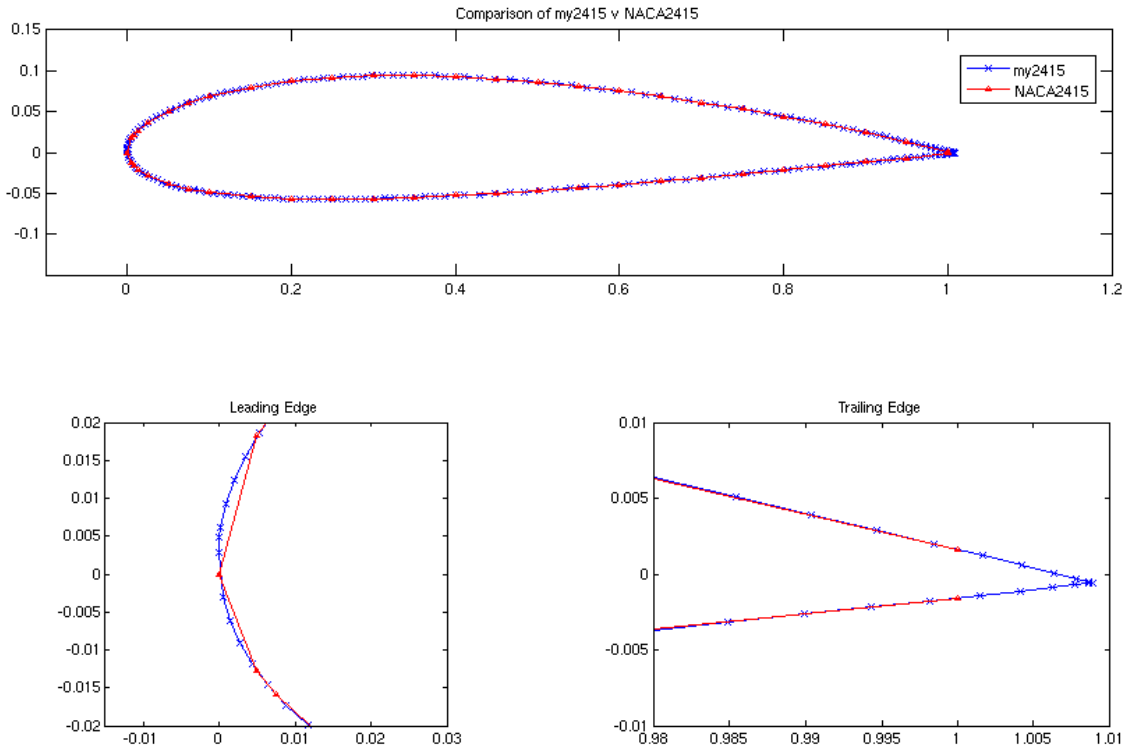


Figure 4.3 Comparisons of Abbott and Von Doenhoff with streamlined 2415.

4.3 Grid Convergence Study

The physics governing fluid dynamics are partial differential equations, using discrete approximations to introduces truncation error. A grid convergence can verify that the discretisation methods used are suitable, and result in a solution that is comparable to published results. It was decided that it is not feasible to conduct a three-dimensional grid

convergence study due to time and resource restrictions, a two-dimensional study was used to guide the mesh generation for the three-dimensional simulation.

Abbott and Von Doenhoff[10] contain the Summary of Airfoil Data[19], which contains velocity profiles over various NACA thickness distributions, mean lines and airfoils. These analytical results determined from conformal mapping, are used as the comparison. The velocity profile determined from flow visualisation over a symmetrical airfoil, NACA0012, and a cambered airfoil NACA2412, and results compared with the published data. The following grid discretisation schemes solved for the flow over a NACA0012 airfoil:

Table 4.1 NACA0012 Grid discretisation schemes.

Label	**GfsRefine**	**GfsSolidRefine**	**No. Cells**	**Timesteps**
Ref0	4	by SolidCurvature from 8:10	1719	11000
Ref1	4	by SolidCurvature from 9:11	2195	21998
Ref2	4	x < 0.025 ? 12 : x > 0.195 ? 12 : 11	5951	62001
Ref3	4	x < 0.025 ? 10 : x > 0.195 ? 14 : 9	3475	181205

Only the GfsSolidRefine command was altered, which subsequently determined the number of cells. In all cases, the flow is resolved for 10 time units; hence number of timesteps needed is an indicator of the refinement of the grid. This is due to the Courant-Friedrichs-Lewy stability condition, where the maximum timestep is limited to prevent overflow of any cell.

Table 4.2 NACA2412 Grid discretisation schemes

Label	GfsRefine	GfsSolidRefine	No. Cells	Timesteps
Ref4	4	by SolidCurvature from 8:10	1731	12008
Ref5	4	by SolidCurvature from 10:12	3243	38331
Ref6	4	10	5951	62001
Ref7‡	4‡	SolidCurvature from 8 : 10 & Adaptation‡	1131‡	12007‡

As mentioned previously, the velocity profile is the property that is compared to the published data. The figure below, illustrates the velocity profiles over the NACA 0012 airfoil for the various discretisation schemes.

‡ Due to Adaptive Grid Refinement, the cell count given is the average. The timestep count is not a useful measure due to the changing grid.

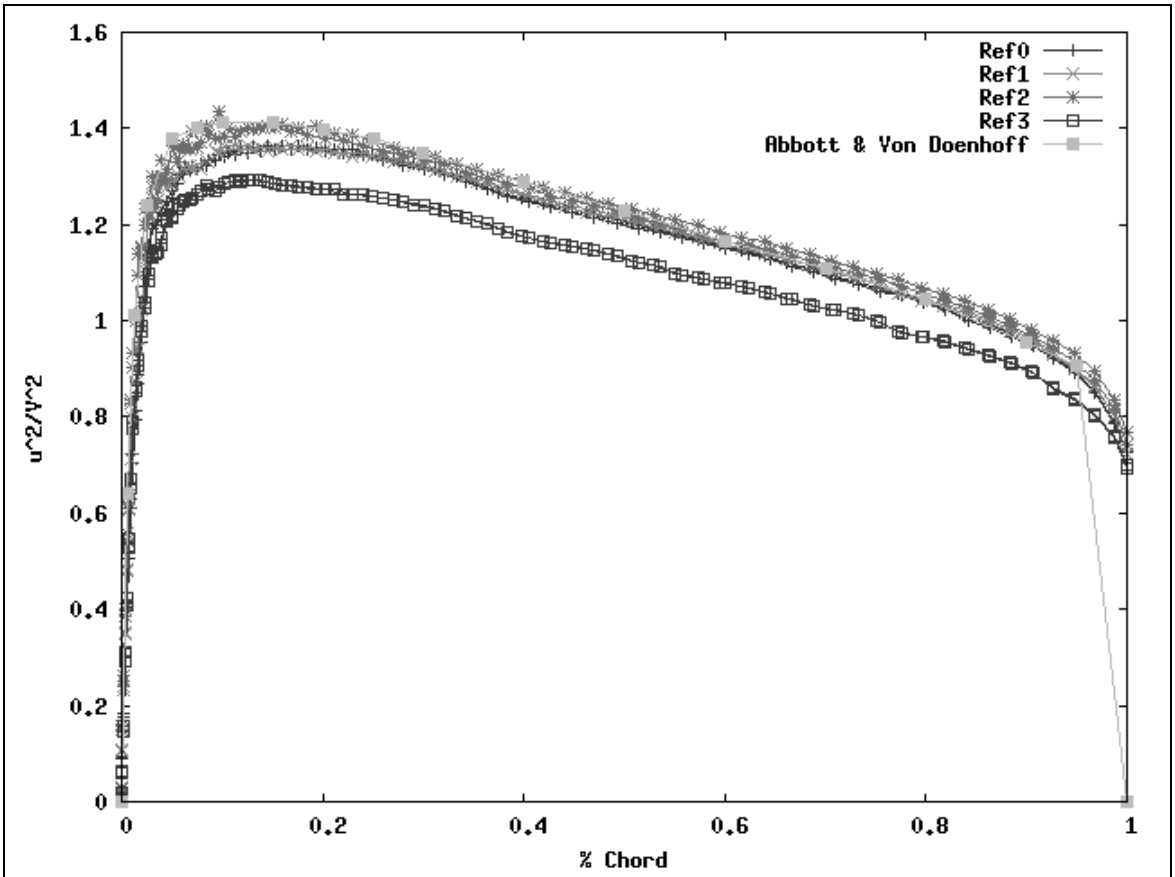


Figure 4.4 Comparison of the velocity profiles over the NACA0012 airfoil

As discussed in the previous section the Kutta-Joukowsky condition is not resolved, hence there is a non-zero velocity at the trailing edge. Apart from the *Ref3* discretisation the Gerris solution corresponds well with the results obtained from Abbott and Von Doenhoff. Also the superfluous vorticity problem, mentioned in section 4.1, has no effect on the resultant flow; hence it may be a visualization problem.

The result of the grid convergence study for the NACA2412 airfoil is displayed in Figure 4.5. There is again a issue with the flow not being smooth at the trailing edge. The *Ref5* and *Ref7* mesh generation schemes produce a velocity profile that matches closely the upper surface velocity profile from Abbott and Von Doenhoff.

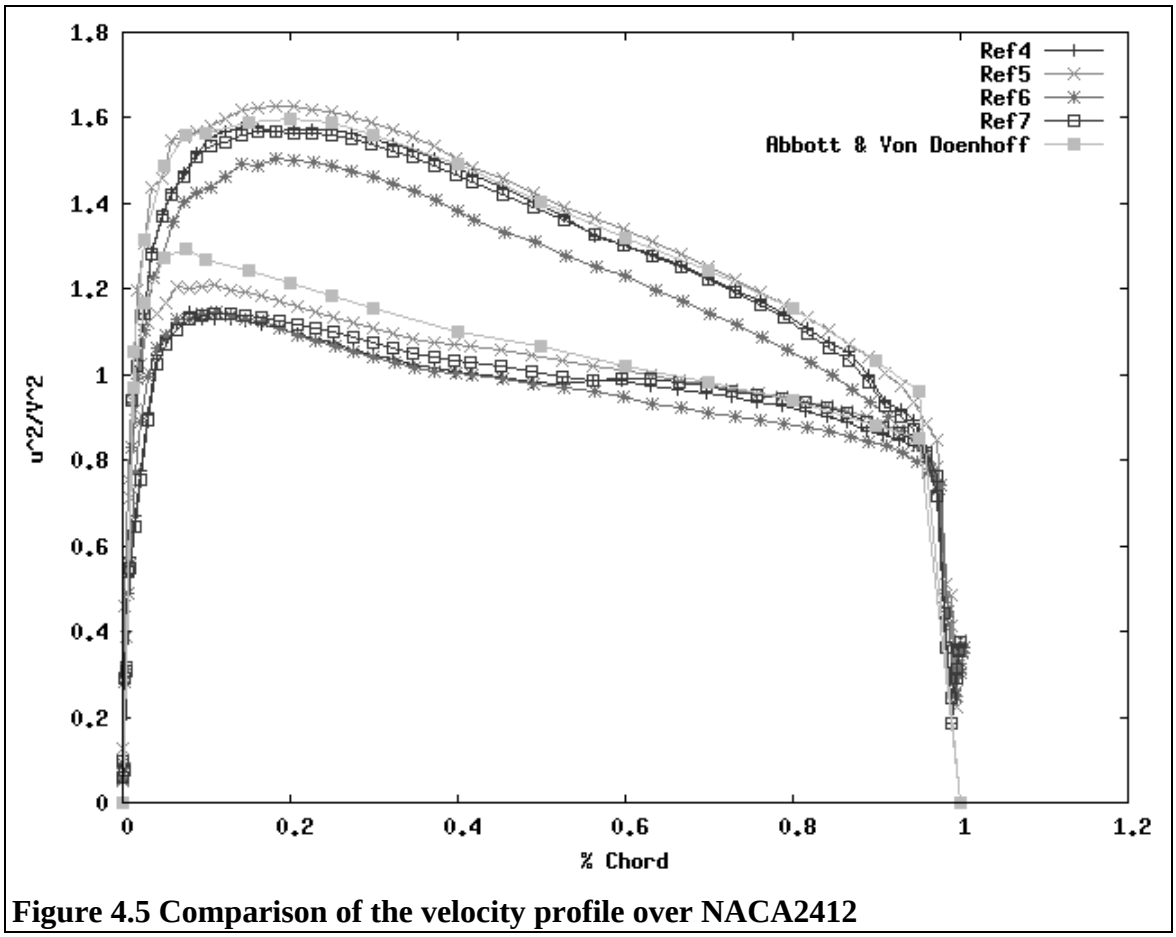


Figure 4.5 Comparison of the velocity profile over NACA2412

5.0 Results

The initial condition of (U=1) uniform translatory motion, results in a discontinuity at the sharp trailing edge, as observed by Prandtl[9]. The discontinuity rolls up into the starting vortex, and is washed downstream. This starting vortex at t = 0.8 can be seen in Figure 5.1, the contours of the vorticity field.

Figure 5.1 Vorticity field, showing the starting vortex at t = 0.8

This flow feature is washed downstream, and departs the domain of analysis at approximately 2.5 time units. A steady flow is resultant, the high pressure and lower pressure flow mixing leads to the formation of the tip vortex. The pressure contours about the wing is illustrated below. This corresponds well to Mönttinen's Figure 5.65.

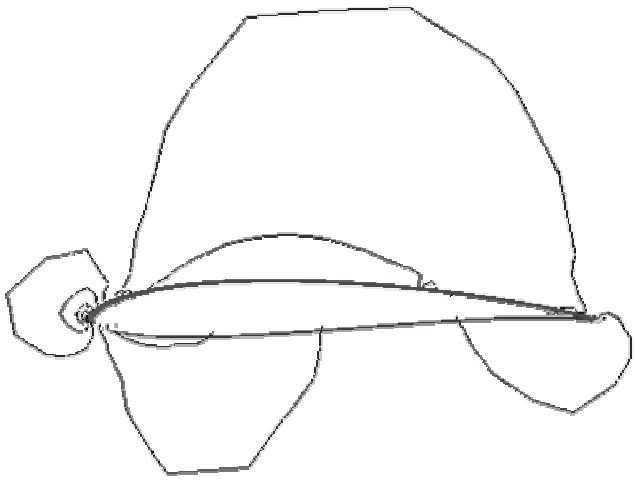


Figure 5.2 Pressure contours at 80% span on base wing.

The effect of the winglet on the circulation about the airfoil can be observed by looking at the pressure distribution about the airfoil at various stations along the semi-span. The stations used are illustrated in Figure 5.2 below.

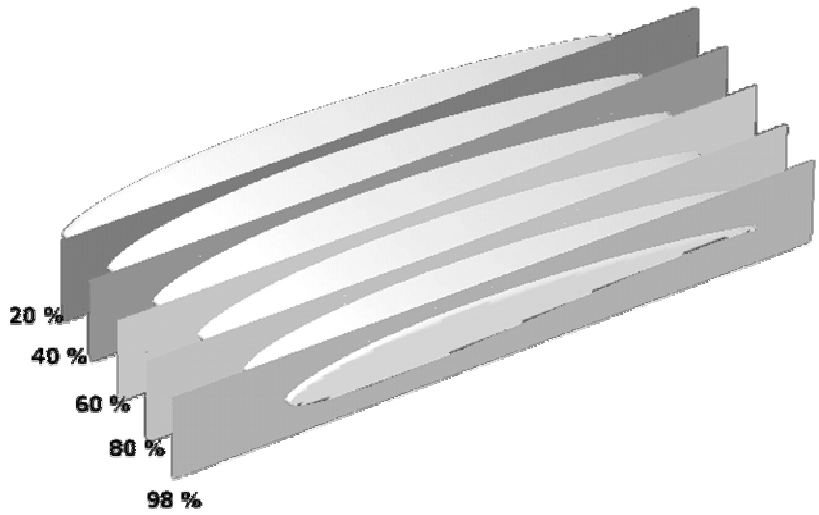


Figure 5.3 Semispan locations along Base wing.

The comparison of winglet wA model with the base wing is illustrated in Figure 5.4, the winglet wA model has higher pressure distribution throughout all the z stations. The difference is most notable towards tips of the wings. The spanwise variation of pressure for the winglet wB model is shown in Figure 5.5, in comparison to the base wing. The wB winglet model has a pressure distribution similar to the base wing in the inboard stations of 20, 40 and 60 percent span.

The lower surface distribution matches very closely to the base wing up to even 80 percent span. The effect of the winglets is most drastic at the 98 percent span station. The peaks and troughs in the pressure distribution of the base wing at the same station have been replaced by a fairly smooth low pressure.

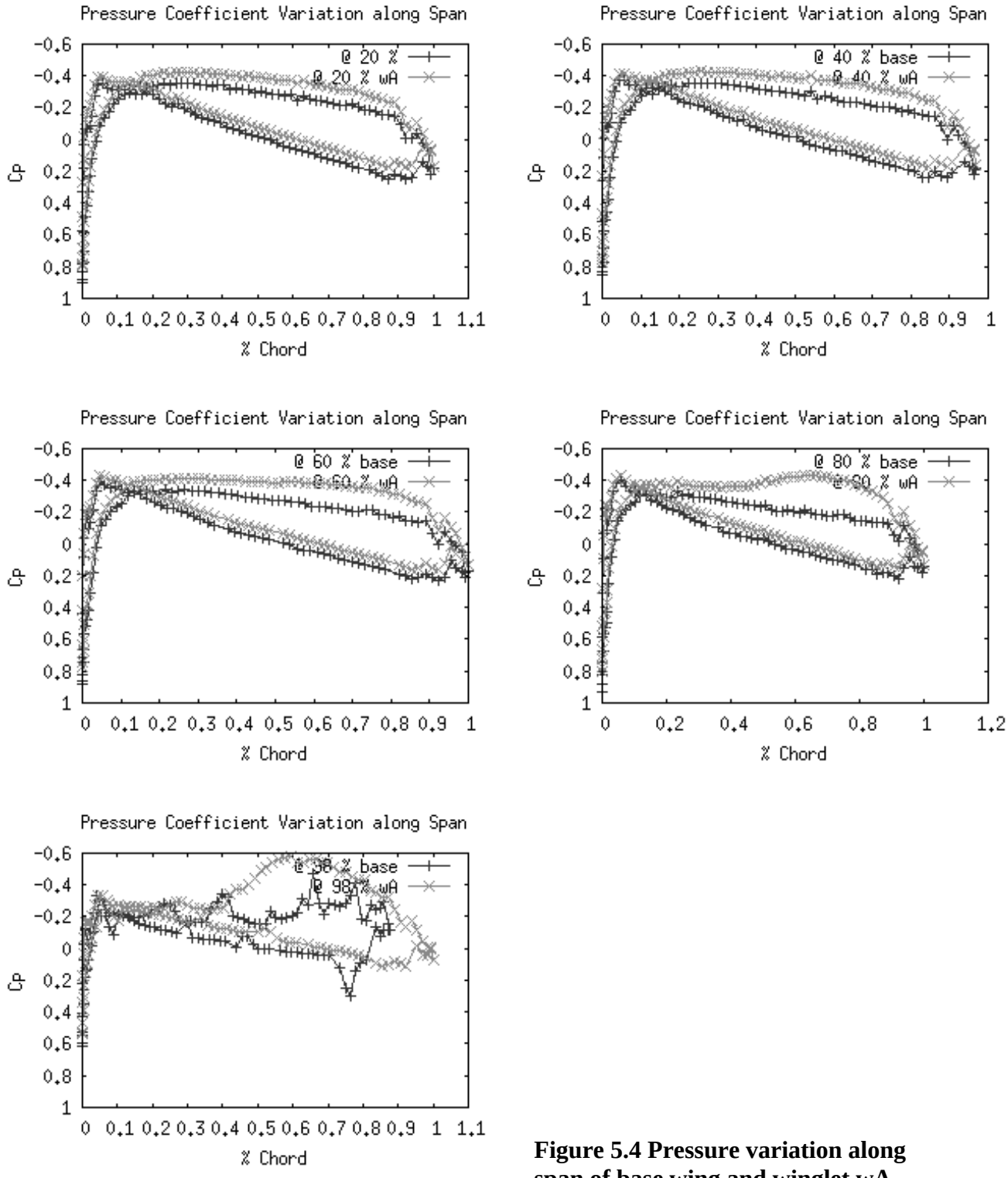
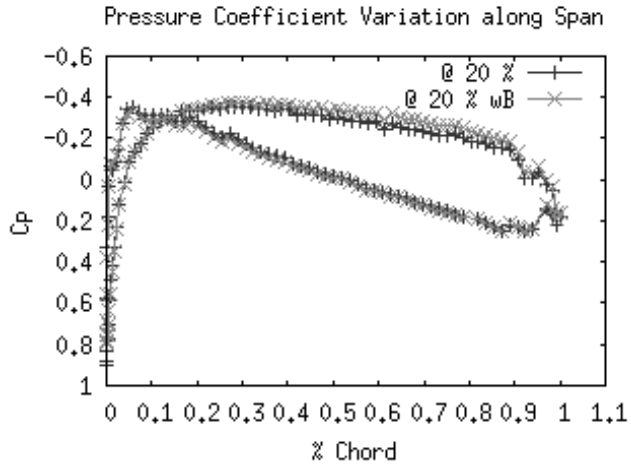
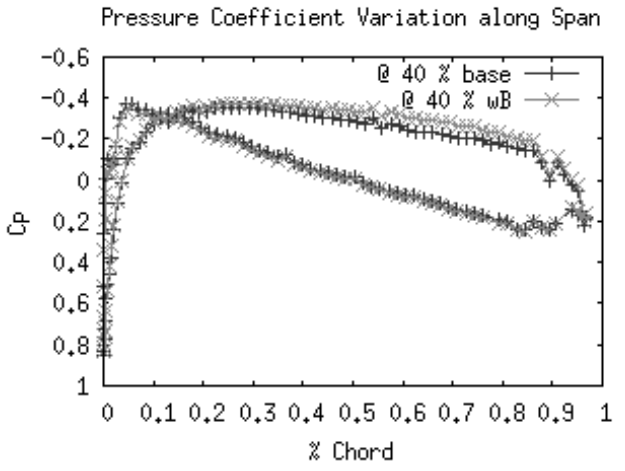
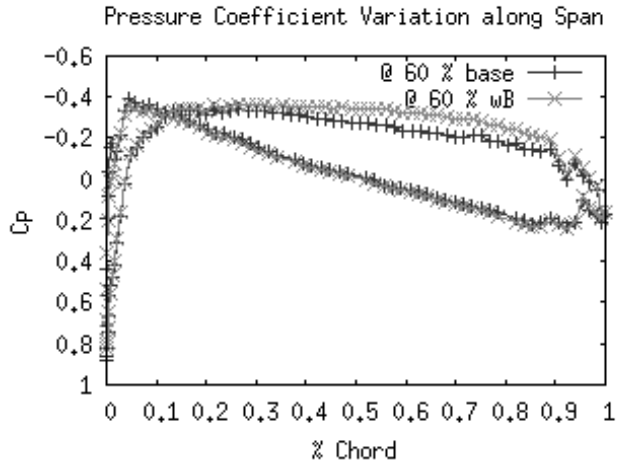
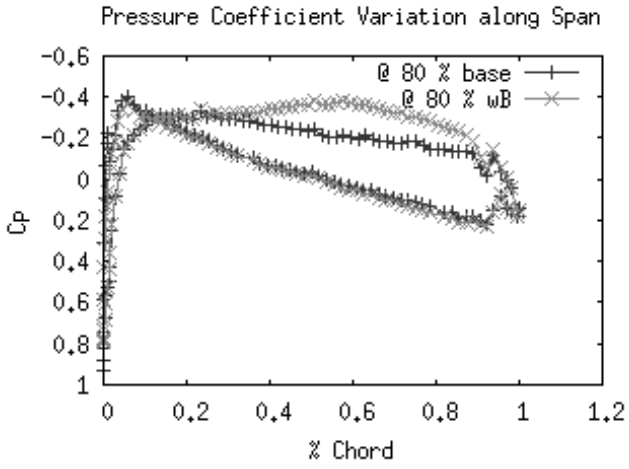


Figure 5.4 Pressure variation along span of base wing and winglet wA.









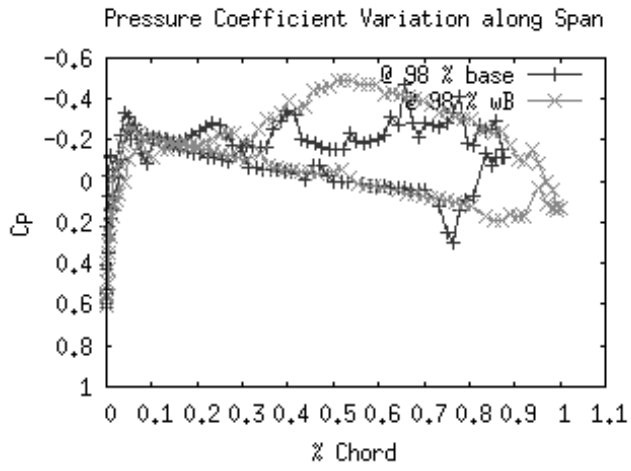


Figure 5.5 Pressure variations along span of winglet wB.

The rollup of the tip vortex is highlighted by streamlines flowing over the wing; the following are screenshots from GfsView where a sheet of streamlines had been placed over the wing

.

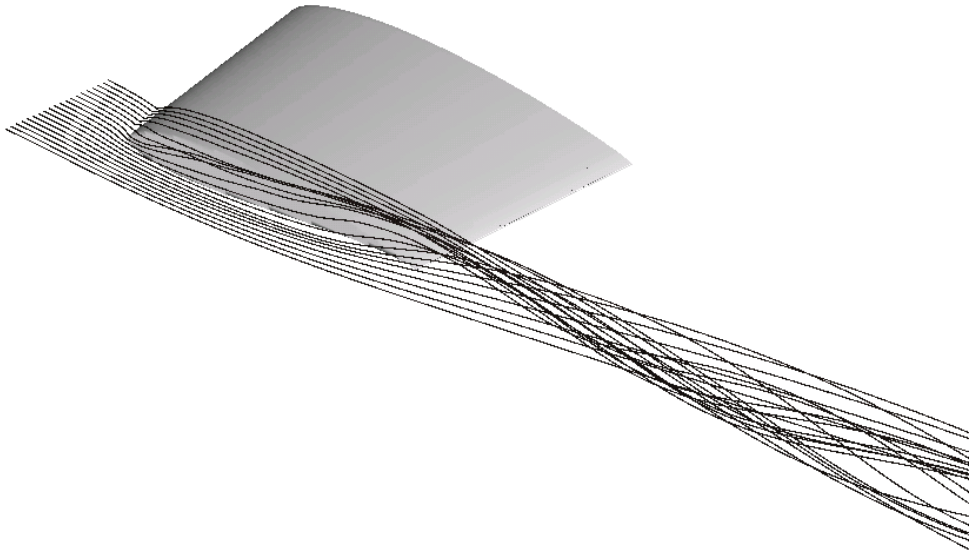


Figure 5.6 Rollup of tip vortex on base wing

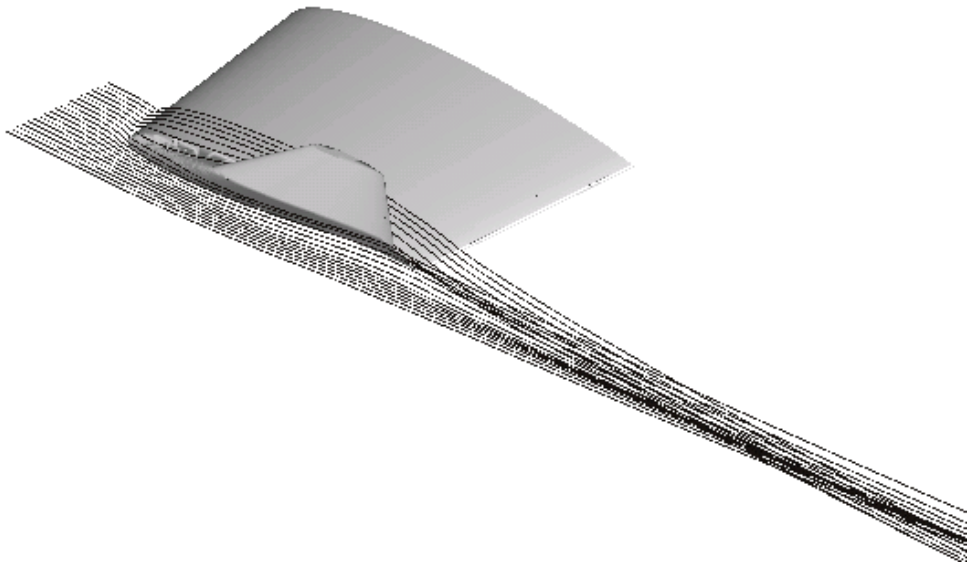


Figure 5.7 Rollup of tip vortex on wA model

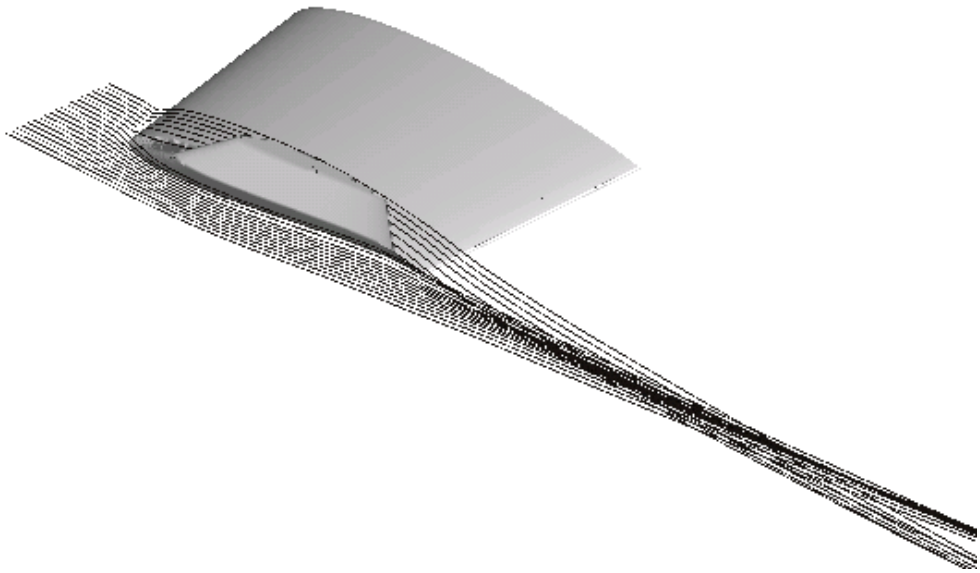


Figure 5.8 Rollup of tip vortex on wB model

There is significant spiraling of the streamlines over the base wing, as shown in Figure 5.6; this can be a guide to the strength of the vortex. Figure 5.9 is a Treftz plane, one chord downstream of the solid looking in the upstream direction. This illustrates the high circulation.

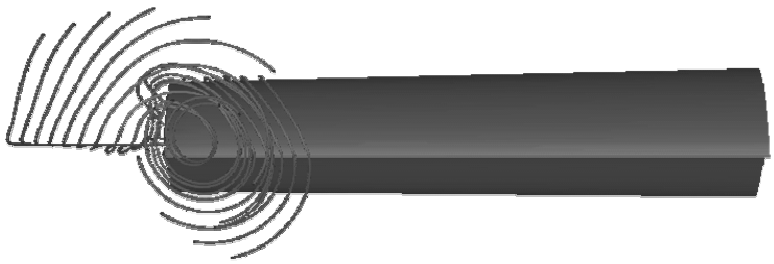


Figure 5.9 Treftz plane section of tip vortex on base wing

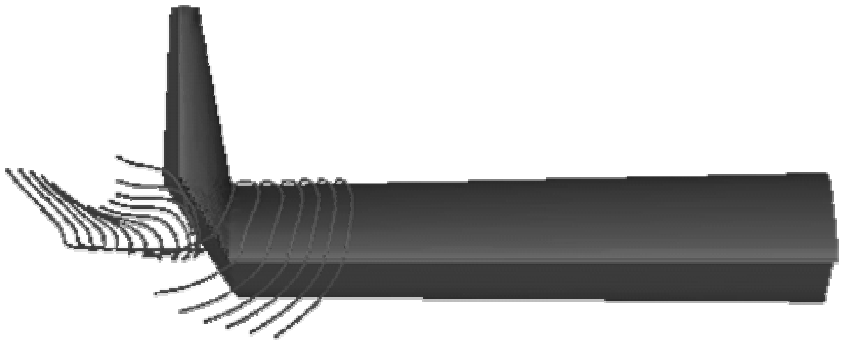


Figure 5.10 Treftz plane section of tip vortex on wA model

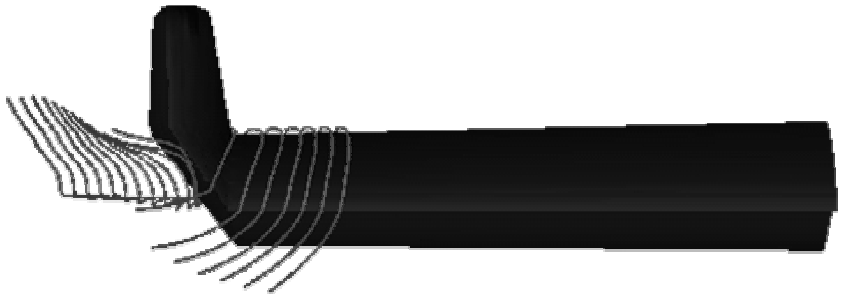


Figure 5.11 Treftz plane section of tip vortex on wB wing

Figures 5.10 and 5.11 are the corresponding treftz plane views for the wA model and wB model respectively. They illustrated the lack of rollup of the tip vortex in comparison to Figure 5.9.

Discussion

The GTSwing library is able to generate three-dimensional wings in the GTS representation, encompassing rectangular, trapezoidal, and elliptical and delta planforms. Almost any wing can be generated given the planform parameters are specified. Addition of cambered winglets is also implemented.

The grid convergence studies, as noted in section 4.2, highlighted the problems arising from using square cells to discretise the fluid domain. These square cells were not able to accurately represent the solid curvature, even with a high level of refinement. The level of refinement that could be used is limited by the time and resources, as the smaller the cell the shorter the timestep due to the CFL stability condition.

Significant errors were observed using the Gerris GfsOutputSolidForce command, with a high level of noise in the output and inconsistent pressure field in symmetric flow. The GfsOutputLocation command was used get the values of pressure and velocity components around the wing section profile.

Results indicate that a winglet will delay the rollup of the tip vortex, and hence reduce its effect on the pressure distribution over the wing.

Further work can be conducted in the viscid flow visualisation, angle of attack variation of tip vortex and extending the GTSwing library.

References

[1]McMichael, J. M., and Francis. M. S., "Micro Air Vehicles – Toward a New Dimension in Flight," DARPA, USA, 1997[§]

[2]Viieru, D., Albertini R., Shyy, W., Ifju, P., "Effect of Tip Vortex on Wing Aerodynamics of Micro Air Vehicles," *22nd Applied Aerodynamics Conference and Exhibit*, Providence, Rhode Island, Aug. 16-19, 2004.

[3]Mönttinen, J., "Computational Study on the Effect of Winglets on the Performance of Micro Air Vehicles," *Postgraduate Thesis*, Arizona State University, 2004.

[4]Tietjens, O. G., and Prandtl, L. *Applied Hydro- and Aeromechanics*, Dover, New York, 1957.

[5]Popinet, S., Gerris: "A Tree-based Adaptive Solver for the Incompressible Euler Equations in Complex Geometries," *Journal of Computational Physics*, Vol. 190, No. 2, 2003, pp. 572-600.

[6]Pelletier, A., and Meuller, T., "Low Reynolds Number Aerodynamics of Low-Aspect-Ratio, Thin/Flat/Cambered-Plate Wings," *Journal Of Aircraft*, Vol. 37, No. 5, 2000, pp. 825-832.

[7]Selig, M. S., Donovan, J. F., and Fraser, D. B., *Airfoils at Low Speeds*, Stokely, Virginia Beach, USA, 1989.

[8]Spoerry, T., and Wong K. C., "Design and Development of a Micro Air Vehicle Concept: Project Bidule," University of Sydney, 1998.

[9]Prandtl, L., *Fundamentals of Hydro and Aeromechanics*, 1934, Dover, New York.

[10]Abbott, I. H., and Von Doenhoff, A. E., Theory of Wing Sections, 1959, Dover, New York.

[11]Tang, Jian, and Zhu Ke-Qin, "Numerical and Experimental Study of Flow Structure of Low-Aspect-Ratio Wing," *Journal of Aircraft*, Vol. 41, No. 5, 2004, pp. 1196-1201.

[12]Cosyn, P., and Vierendeels, J., "Numerical Investigations of Low Aspect Ratio Wings at Low Reynolds Numbers," 23rd AIAA Applied Aerodynamics Conference and Exhibit, Toronto, Ontario, June 6-9, 2005, AIAA-2005-4609.

[§] This presentation is difficult to find. As of 8/8/2006 it was available at http://uler.aero.iitb.ac.in/docs/MAV/www.darpa.mil/tto/MAV/mav_ausi.html

[13]Torres, G. E., and Meuller, T. J., "Low-Aspect-Ratio Wing Aerodynamics at Low Reynolds Numbers," *AIAA Journal*, Vol. 42, No. 5, 2004, pp. 865-873.

[14]Sathaye, S. S., "Lift Distributions on Low Aspect Ratio Wings at Low Reynolds Numbers", *Masters Thesis*, Worcester Polytechnic Institute, 2004.

[15]Carmichael. B. H., "Low Reynolds Number Airfoil Survey Volume 1", *NASA Contractor Report 165803*, Jan. 1982.

[16]Zimmerman, C. H., "Aerodynamic Characteristics of Several Airfoils of Low Aspect Ratio", *NACA Technical Note 539*, Aug. 1935.

[17]Mueller, T. J., and DeLaurier, J. D., "Aerodynamics of Small Vehicles," *Annual Review of Fluid Mechanics*, 2003, 35:89-111.

[18]Blender, http://www.blender.org , 6/6/06.

[19]Abbott, I. H., Von Doenhoff, A. E., Stivers, Jr. L. S., "Summary of Airfoil Data", 1933, NACA Report No. 824**

[20]Eppler, R., Airfoil Design and Data, 1990, Springer-Verlag, Berlin.

[21]Nihon University Aero Student Group Airfoil Database, http://www.nasg.com/afdb

** This report is incorporated into Theory of Wing Sections[10]

Appendix A - GTSWing-0.0.22 Library for Matlab/Octave

Defining the physical domain is an important step in CFD. The boundary conditions, fluid properties, initial conditions need to be defined appropriately. When modeling flow around solid objects, such as in wall-bounded flow, the solid has to be represented in a way that the solver would understand. Gerris has the ability to read surfaces in the GTS format. Simple objects can be converted into GTS format with little issue, however when dealing with complex objects with non-planar surfaces it may become difficult. In the initial stages of this analysis the need for a triangulation script became apparent. As the solid to be represented grew in complexity, by the addition of winglets and non-planar planform shapes, this triangulation script was replaced by a library of functions. Developed in a modular way, these functions greatly enhanced the functionality and enable complex shapes to be converted to the GTS format. These functions are packaged as the GTSWing library. They consist of the following:

Planform Shape Functions

Function : mep

```
function [chord]=mep(aspect_ratio, tip_root_chord, spanwise_loc)
%
% Modified Elliptical Planform
%
% Returns the chord length, at a given spanwise location
%
% tip_root_chord = chord at tip/ chord at root
% spanwise_loc = spanwise station (0 - 1)
%
% Sujee Mampitiyarachchi, 30th June 2006

% Tip to Root chord ratio

TIPCHORD = 1*tip_root_chord;
```

```
if spanwise_loc == 0 || spanwise_loc == 1,
    chord = TIPCHORD;
    return;
end

% Ellipse of AR = 0.75
b = 0.5*(1 - TIPCHORD);
a = aspect_ratio/2;

chord = TIPCHORD+ 2*b*sqrt(1-(spanwise_loc*aspect_ratio-
a)*(spanwise_loc*aspect_ratio-a)/a/a);
```

Function : delta

```
function [chord] = delta(aspect_ratio, sweep, spanwise_loc)
% [chord] = delta(aspect_ratio, sweep, spanwise_loc)
%
% Delta Planform shape function
%
% aspect_ratio - full span aspect ratio
% sweep - leading edge sweep
% spanwise_loc - location along span (0 - 1)
%
% Sujee Mampitiyarachchi, 13/09/06

% assertion
if ~(aspect_ratio/2*tan(sweep)<1),
    display('**ERROR assertion aspect_ratio/2*tan(sweep)<1 failed.
SWEEP is too high');
    return;
end

if spanwise_loc == 0.5,
    chord = 1;
    return
end

if spanwise_loc > 0.5
    chord = 1 - 2*(spanwise_loc*aspect_ratio/2 - 0.5)*tan(sweep);
else
    chord = 1 - 1*(1-2*spanwise_loc*aspect_ratio/2)*tan(sweep);
end
```

Function : trp

```
function [chord]=trp(TIP_ROOT, spanwise_loc)
% [chord]=trp(TIP_ROOT, spanwise_loc)
%
% Tapered Rectangular Planform
%

% Root chord
if spanwise_loc == 0.5,
```

```
    chord = 1;
    return;
end

if spanwise_loc > 0.5,
    chord = (TIP_ROOT-1)/0.5*(spanwise_loc-0.5) + 1;
else
   chord = (1-TIP_ROOT)/0.5*spanwise_loc +TIP_ROOT;
end
```

Discretization Functions

Function : epwing

```
function [] = epwing(airfoil_data,AR, SECTION, TIP_ROOT, NOS)
% [] = epwing(airfoil_data,AR, SECTION, TIP_ROOT, NOS)
%
% Constructs a GNU Triangulated Surface Representation of a
% Elliptical Planform Wing.
%
% airfoil_data -  matrix of 2 columns, [x,y], coordinates defining
%                 the airfoil section.
%
% AR - aspect ratio
% SECTION - 1 for closed section at wingtips,
%           0 for open section at wingtips.
% TIP_ROOT - ratio of tip chord length over root chord length
% NOS - number of segments to discretise the wing into, must be > 1.
%
% A file is outputed called epwing.gts which contains
% the GTS surface of the Elliptical Planform Wing.
%
% Sujee Mampitiyarachchi,11/09/06


OFNAME = 'epwing.gts';

SPAN=AR;

% Assertion
if ~(NOS > 1),
    display('**ERROR: assertion ( NOS > 1 ) failed')
    return;
end

% Checking for inconsistent input parameters
if TIP_ROOT == 0,
    SECTION = 0;
end

% Distribute z-stations sinusoidally.
zstations =[-1:2/NOS:1]';
zstations = sin(pi*zstations/2);
```

```
zstations = (zstations+1)/2;

for i=1:NOS+1,
    chords(i)=mep(SPAN, TIP_ROOT, zstations(i));
    if i~=1,
        span(i-1)=SPAN*(zstations(i)-zstations(i-1));
        sweep(i-1)=atan((chords(i)-chords(i-1))/2/span(i-1));
        z_offset(i-1)=0;
        x_offset(i-1)=0;
        taper1(i-1)=chords(i-1);
        taper2(i-1)=chords(i);
    end

    if i>2,
        x_offset(i-1)=x_offset(i-2)-span(i-2)*tan(sweep(i-2));
        z_offset(i-1)=z_offset(i-2)+span(i-2);
    end
end

buildwing(OFNAME, airfoil_data, TIP_ROOT, x_offset, z_offset, sweep,
span, taper1, taper2, SECTION);
```

Function : dpwing

```
function [] = dpwing(airfoil_data, AR, SECTION, SWEEP, NOS)
% [] = epwing(airfoil_data,AR, SECTION, SWEEP, NOS)
%
% Constructs a GNU Triangulated Surface Representation of a
% Delta Planform Wing.
%
% airfoil_data -  matrix of 2 columns, [x,y], coordinates defining
%                 the airfoil section.
% AR - aspect ratio
% SECTION - 1 for closed section at wingtips,
%           0 for open section at wingtips.
% SWEEP - sweep of leading edge, (0-pi/2)
% NOS - number of segments to discretise the wing into.
%
% A file is outputed called epwing.gts which contains
% the GTS surface of the Delta Planform Wing.
%
% Sujee Mampitiyarachchi,13/09/06

OFNAME = 'dpwing.gts';

SPAN=AR;

if ~(SWEEP>0 && SWEEP<pi/2),
    display('**ERROR: assertion ( SWEEP>0 && SWEEP<pi/2) failed')
end

zstations =[-1:2/NOS:1]';
zstations = sin(pi*zstations/2);
zstations = (zstations+1)/2;
```

```
for i=1:NOS+1,
    chords(i)=delta(AR,SWEEP, zstations(i));
    if i~=1,
        span(i-1)=SPAN*(zstations(i)-zstations(i-1));
        sweep(i-1)=SWEEP*(chords(i)-chords(i-1))/abs(chords(i)-
chords(i-1));
        z_offset(i-1)=0;
        x_offset(i-1)=0;
        taper1(i-1)=chords(i-1);
        taper2(i-1)=chords(i);
    end

    if i>2,
        x_offset(i-1)=x_offset(i-2)-span(i-2)*tan(sweep(i-2));
        z_offset(i-1)=z_offset(i-2)+span(i-2);
    end
end

TIP_ROOT=chords(1);

buildwing(OFNAME, airfoil_data, TIP_ROOT, x_offset, z_offset, sweep,
span, taper1, taper2, SECTION);
```

Function : rpwing

```
function [] = rpwing(airfoil_data, AR, SECTION,TIP_ROOT, SWEEP, TWIST,
DIHEDRAL, NOS)
% [] = rpwing(airfoil_data, AR, SECTION, TIP_ROOT, SWEEP, TWIST,
DIHEDRAL, NOS)
%
% Constructs a GNU Triangulated Surface Representation of a
% Rectangular Planform Wing.
%
% airfoil_data -  matrix of 2 columns, [x,y], coordinates defining
%                 the airfoil section.
% AR - aspect ratio
% SECTION - 1 for closed section at wingtips,
%           0 for open section at wingtips.
% TIP_ROOT - ratio of tip chord length over root chord length
% SWEEP - sweep of leading edge, (0-pi/2)
% TWIST - twist (constant)
% DIHEDRAL - dihedral (constant)
% NOS - number of segments to discretise the wing into.
%
% A file is outputed called rpwing.gts which contains
% the GTS surface of the Rectangular Planform Wing.
%
% Sujee Mampitiyarachchi,19/09/06

OFNAME = 'rpwing.gts';

SPAN=AR;

% Assertion
if ~(NOS > 1),
```

```
    disp('**ERROR: assertion ( NOS > 1 ) failed')
    return;
end

if ~(SWEEP>=0 && SWEEP<pi/2),
    disp('**ERROR: assertion ( SWEEP>0 && SWEEP<pi/2) failed')
    return;
end

if TIP_ROOT == 0,
    SECTION = 0;
end

zstations =[-1:2/NOS:1]';
zstations = sin(pi*zstations/2);
zstations = (zstations+1)/2;

for i=1:NOS+1,
    chords(i)=trp(TIP_ROOT, zstations(i));
    if i~=1,
        span(i-1)=SPAN*(zstations(i)-zstations(i-1));

      if SWEEP == 0
        sweep(i-1)=0;
      else
        sweep(i-1)=SWEEP*(chords(i)-chords(i-1))/abs(chords(i)-
chords(i-1));
      end

        z_offset(i-1)=0;
        x_offset(i-1)=0;
        taper1(i-1)=chords(i-1);
        taper2(i-1)=chords(i);
    end

    if i>2,
        x_offset(i-1)=x_offset(i-2)-span(i-2)*tan(sweep(i-2));
        z_offset(i-1)=z_offset(i-2)+span(i-2);
    end
end

buildwing(OFNAME, airfoil_data, TIP_ROOT, x_offset, z_offset, sweep,
span, taper1, taper2, SECTION);
```

Function : tsst

```
function [points, lines, tris] =tsst(profile1, profile2, ends)
%[points, lines, tri] =tsst( profile1, profile2, ends)
% Tapered, Swept Segment Triangulation function
%
% Triangulates the Surface of a segment of a wing, and returns in gts
% representation.
%
% profile1 = coordinates specifying the first section profile
% profile2 = coordinates specifying the second section profile
```

```
% ends = whether the ends are on.
%           -1 the first profile is meshed
%           1 the 2nd profile is meshed
%           0 neither profile is meshed
%           2 both profiles are meshed
%
% Sujee Mampitiyarachchi, 3/8/06

profile2=round(1e6*profile2)/1e6;
profile1=round(1e6*profile1)/1e6;

if profile1(:,[1 2])==zeros(length(profile1),2),
    [p3,l3,t3]=gtsEx2Vert(profile2,profile1(1,:));
    t3=[t3(:,3), t3(:,2), t3(:,1)];
    ends=0;
else
    if profile2(:,[1 2])==zeros(length(profile2),2),
        [p3,l3,t3]=gtsEx2Vert(profile1, profile2(1,:));
        ends=0;
    else
        NoP = length(profile1); % Total number of Points

        % 1st Profile
        [p,l,t]=gts2D(profile1(:,1),profile1(:,2));
        p=[p(:,1), p(:,2), profile1(:,3)];
        t=[t(:,3), t(:,2), t(:,1)];

        % 2nd Profile
        [p2,l2,t2]=gts2D(profile2(:,1),profile2(:,2));
        p2=[p2(:,1), p2(:,2), profile2(:,3)];

        %%%%%%%%%%%%%%%%%%%%%%%%%
        % 1-2 Profile
        [p3,l3,t3]=gtsExtrude(profile1, profile2);
    end
end

% Join the 3 Surfaces together
if ends == 0,
    points=p3;    lines=l3;    tris=t3;
elseif ends == -1,
        [points, lines ,tris] = gtsmerge(p, l, t, p3, l3, t3);
    elseif ends == 1,
            [points, lines, tris] = gtsmerge(p2, l2, t2, p3, l3, t3);
            elseif ends ==2,
           [points, lines, tris] = gtsmerge(p,l,t, p2, l2, t2);
            [points, lines, tris] = gtsmerge(points, lines, tris, p3,
l3, t3);
end
```

Function : buildwing

```
function [] = buildwing(OFNAME, airfoil_data, TIP_ROOT, x_offset,
z_offset, ...
    sweep, span, taper1, taper2, SECTION)
```

```
% [] = buildwing(OFHEADER, airfoil_data, TIP_ROOT, x_offset, z_offset,
sweep,
%                                        span, taper1, taper2, SECTION)
%
% Build Wing From Segments
%
% OFNAME - output file name
% airfoil_data  -  matrix of 2 columns, [x,y], coordinates defining
%                 the airfoil section.
% TIP_ROOT - ratio of tip chord length over root chord length
% x_offset - col vector specifying a x_offset for each segment
% z_offset - col vector specifying a z_offset for each segment
% sweep - col vector specifying a sweep for each segment
% span - col vector specifying a span for each segment
% taper1 - col vector specifying the chordlength of the portside
section of a
% segment over the root chord
% taper2 - col vector specifying the chordlength of the starboardside
section of a
% segment over the root chord
% SECTION - 1 for closed wingtip sections
%           0 for open wingtip sections
%
% Sujee Mampitiyarachchi, 14/9/06
HEADER = 'panel';
EXT = '.gts';

x=airfoil_data(:,1);
y=airfoil_data(:,2);
NoP=length(airfoil_data);
NOS = length(sweep);

if ~(SECTION==1 || SECTION==0),
    display('**ERROR: assertion SECTION==1 || SECTION==0 failed');
    return;
end

if
~((length(x_offset)==length(z_offset))&&(length(z_offset)==length(taper
1))...

&&(length(taper1)==length(taper2))&&(length(taper2)==length(span))...
    &&(length(span)==NOS)),
    display(strcat('**ERROR: assertion
(length(x_offset)==length(z_offset))&&',...

'(length(z_offset)==length(taper1))&&(length(taper1)==length(taper2))',
...
     '&&(length(taper2)==length(span))&&(length(span)==NOS) failed'));
    return;
end

if ~(TIP_ROOT > 0),
    display('**ERROR: assertion failed TIP_ROOT > 0');
    return;
end
```

```
ends=zeros(NOS,1);
ends(1)= -SECTION;
ends(length(ends))=SECTION;

% Triangulate segments
segments = length(sweep);

for j=1:segments,

    ofname = strcat(HEADER, num2str(j));
    ofname = strcat(ofname, EXT);

    x_off= x_offset(j)*ones(NoP,1);
    z_off= z_offset(j)*ones(NoP,1);
    sweep_offset = -1*span(j)*tan(sweep(j))*ones(NoP,1);
    Z1=span(j)*ones(NoP,1);

    if (TIP_ROOT == 0 && (j==1 || j==NOS)),
        if(j==1)
            [points,lines,tri] =
gtsEx2Vert([x_off+sweep_offset+taper2(j)*x,taper2(j)*y,
z_off+Z1],[x_offset(j),0,z_off(j)]);
            tri=[tri(:,3),tri(:,2),tri(:,1)];
        else
            [points,lines,tri] =
gtsEx2Vert([x_off+taper1(j)*x,taper1(j)*y,
z_off],[x_offset(j)+sweep_offset(1),0,z_off(j)+Z1(1)]);
        end
    else
        [points, lines, tri]= tsst( [x_off+taper1(j)*x,taper1(j)*y,
z_off], [x_off+sweep_offset+taper2(j)*x,taper2(j)*y, z_off+Z1],
ends(j));
    end

    % Save gts representation
    gtssave(ofname, points, lines, tri);

end

% Merge all the segments
gtsmergeall(OFNAME,HEADER, EXT, NOS);
```

Triangulation Functions

Function : gtsExtrude

```
function [points,lines,tris]=gtsExtrude(profile1, profile2)
% [points,lines,tris]=gtsExtrude(profile1, profile2)
%
% Triangulate a surface from profile1 to profile2
%
% Where profile1 is [x,y,z] forming a airfoil section
%       profile2 is [x,y,z] forming another airfoil section
```

```
%
% length(profile1) == length(profile2)
% No duplicate points in either profile.
% Profile cannot be intersecting
%
% Sujee Mampitiyarachchi, 3/8/06

% 1-2 Profile

points=[profile1;profile2];

NoP = floor(length(points)/2);

j=[1:NoP-1]';

% Bounding Lines of Profiles 1 & 2
lines=[j,j+1; % 1st Profile
    NoP, 1;
    NoP+j, NoP+j+1; % 2nd Profile
    2*NoP, NoP+1];

r=[1:NoP-1]';

% Lines from 1 to 2
lines=[lines;
    r,NoP+r;% Orthogonal
    NoP, 2*NoP;
    r+1, NoP+r; % Diagonal
    1, 2*NoP];

% Creating triangles on surface between profiles.

s=[1:NoP-1]';

tris=[s, 3*NoP+s,2*NoP+s; % [adj, dia, ortho]
    NoP, 3*NoP+NoP, 2*NoP+NoP;
    2*NoP+s+1, NoP+s, 3*NoP+s; % [ortho, adj, dia]
    2*NoP+1, 2*NoP, 4*NoP];
```

Function : gtsEx2Vert

```
function [points, lines, tris] = gtsEx2Vert(profile, vertex)
% [p,l,t] = gtsEx2Vert(profile, vertex)
%
% Triangulate a surface formed from profile to a single point vertex.
%
%
% Sujee Mampitiyarachchi, 17/8/06

points=[profile;
    vertex];

NoP=length(points);

j=[1:NoP-2]';
```

```
% Bounding lines of profile
lines=[j,j+1;
    NoP-1,1];

% Lines from profile to vertex
lines=[lines;
    j,NoP*ones(length(j),1);
    NoP-1, NoP];

% Triangles
tris=[j,NoP+j,NoP-1+j;
    NoP-1,NoP, 2*NoP-2];
```

Function : gts2D

```
function [points, lines, tris] = gts2D(x,y)
% [points, lines, tris] = gts2D(x,y)
%
% Triangulates a 2D planar surface bounded by the curve specified by x,
y.
%
% Sujee Mampitiyarachchi, 8/08/06

points = [x,y];

NoP = length(points);

% creating lines on each profile
% outside of profile
i=[1:NoP-1]';
lines=[i,i+1;
    NoP, 1];

if (ceil(NoP/2)-floor(NoP/2))==0,
% Triangulating EVEN loops
% lines within profile
c=[1:NoP/2-2]';

%length(c)
lines=[lines;
    c+1, NoP-c; % diagonal
    c, NoP-c; % orthogonal
    NoP/2-1, NoP/2+1];
%length(lines)

t=[1:NoP/2-2]';

tris=[t, NoP+t, 1.5*NoP-2+t; % Lower Triangles [adj,dia,ortho]
    NoP/2-1, NoP/2, 2*NoP-3; % LeftOver Lead Triangle
     NoP/2+t, 1.5*NoP-1-t, 2*NoP-2-t; % Upper Triangles[adj,dia,ortho]
     NoP-1, NoP,1.5*NoP-1]; % LeftOverTrailing Triangle
else

% Triangulating ODD Loops
c=[1:(NoP-3)/2]';
```

```
lines=[lines;
    c+1, NoP-c;
    c, NoP-c];

t=[1:(NoP-3)/2-1]';

tris=[t, NoP+t, NoP+(NoP-3)/2+t;
    (NoP-3)/2, NoP+(NoP-3)/2, 2*NoP-3;
    (NoP-1)/2, (NoP+1)/2, NoP+(NoP-3)/2;
    (NoP+1)/2+t, NoP+(NoP-3)/2-t, 2*NoP-2-t;
    NoP-1,NoP,NoP+(NoP-3)/2+1];

end
```

Function : gtsWinglet

```
function [points, lines, tris] = gtswinglet(profile, taper_ratio,
x_offset, span, sweep,ends)
% [[points, lines, tris] = gtswinglet(profile, taper_ratio, x_offset,
span,
% sweep,ends)
%
% profile - matrix [x,y,z] containing the section profile of wing to
where
% winglet joins
% x_offset - location of leading edge of winglet wrt to leading edge of
% wing tip.
%
% span - height of winglet in terms of wing tip chord
% sweep - sweep of leading edge of winglet
%
% Sujee Mampitiyarachchi, 18/09/06

% Mesh Winglet/Wing Join Section

if ~(x_offset > 0 && x_offset < 1),
    display('**ERROR assertion (x_offset > 0 && x_offset < 1) failed');
    return;
end

if ~(sweep >= 0 && sweep <= 180/pi*atan2(1-x_offset, span)),
    display('**ERROR asertion (sweep >= 0 && sweep <= 180/pi**atan2(1-
x_offset, span)) failed');
    return;
end

wtchord = max(profile(:,1));

tprofile=[profile(:,1)*(1-x_offset),profile(:,2)*(1-
x_offset),profile(:,3)];

tprofile(:,1)=tprofile(:,1)+x_offset*wtchord; % x_offset
[y,I] = min(tprofile(:,1));
dy=max(profile(:,2))-tprofile(I,2);
tprofile(:,2)=tprofile(:,2)+dy*1.05*ones(length(tprofile),1);
```

```
if ends==1,
    w1=gtsRotate(tprofile,pi/2,'x', tprofile(I,:));
    dz= abs(min(w1(:,3))-profile(1,3));
    w1(:,3)=w1(:,3)+1*dz*ones(length(w1),1)+0.02*dy;
end

if ends ==-1,
    w1=gtsRotate(tprofile,-pi/2,'x', tprofile(I,:));
    dz=profile(1,3)-max(w1(:,3));
    w1(:,3)=w1(:,3)+1.0*dz*ones(length(w1),1)-0.02*dy;
end

[jp,jl,jt]=gtsExtrude(profile,w1);

% Taper
[y1,I1]=min(w1(:,1));
w1z=w1(I1,3);
w2=[x_offset*wtchord+taper_ratio*(w1(:,1)-
min(w1(:,1))*ones(length(w1),1)),w1(:,2)+span*wtchord,taper_ratio*(w1(:
,3)-w1z)+w1z];

% Leading edge sweep
if sweep ~= 0,
    w2(:,1)=w2(:,1)+span*wtchord*tan(sweep*pi/180)*ones(length(w2),1);
end

[wp,wl,wt]=gtsExtrude(w1,w2);
[tp, tl, tt]=gtsmerge(wp,wl,wt,jp,jl,jt);
[w3,w3l,w3t]=gts2D(w2(:,1),w2(:,3));
w3=[w3(:,1),w2(:,2),w3(:,2)];

[points,lines,tris]=gtsmerge(tp,tl,tt,w3,w3l,w3t);
```

Merging Functions

Function : gtsmerge

```
function [points, lines, tris] = gtsmerge(p1, l1, t1, p2, l2, t2)
% [points, lines, tris] = gtsmerge(p1, l1, t1, p2, l2, t2)
%
% Join 2 GTS surfaces represented by p1, l1, t1 and p2, l2, t2
%
% Returns [points, lines, tris] representing the merged surface.
%
% Does not recognise intersecting faces.
%
% Sujee Mampitiyarachchi, 14/07/06

p2 = round(1e6*p2)/1e6;
p1 = round(1e6*p1)/1e6;

if length(p2) > length(p1),
```

```
    pt=p1; lt=l1; tt=t1;
    p1=p2; l1=l2; t1=t2;
    p2=pt; l2=lt; t2=tt;
end

nop1=length(p1);
nop2=length(p2);
nol1=length(l1);
nol2=length(l2);
not1=length(t1);
not2=length(t2);

%%%%%%%%%%% POINTS %%%%%%%%%%%%%
% Stores x,y,z coordinates of all pts in merged surface
points = zeros(nop1+nop2,3);
%length(points)

% Mapps pts from 2nd GTS to index of merged pts
p_count=1;

% Read in pts from 1st GTS
points=p1;

p_count = 1+nop1;

% Read in pts from 2nd GTS
for j=1:nop2,
    entry = p2(j,:);
    index = exists(points, entry);

    ptsmap(j)=index;

    if index==0,
        points(p_count,:)=entry;
        ptsmap(j)=p_count;
        p_count=p_count+1;
    end
end

ptsmap=ptsmap';

% Trim points array
points=points([1:p_count-1],:);

%%%%%%%%%% LINES %%%%%%%%%%%%%%%%%
% Stores start and end points of each line
lines = zeros(nol1+nol2, 2);
lmap = zeros(nol2,1);

% Read in lines from 1st GTS
lines = l1;
l_count=1+nol1; % next empty index

% Read in lines from 2nd GTS
for l=1:nol2,
    tmp = l2(l,:);
```

```
    % Map into join reference frame
    entry=ptsmap([l2(l,:)],:)';

    index = exists(lines, entry);
    lmap(l)=index;

    if index==0,
        lines(l_count,:)=entry;
        lmap(l)=l_count;
        l_count=l_count+1;
    end

end

% Trim lines array
lines=lines([1:l_count-1],:);

%%%%%%%%% TRIANGLES %%%%%%%%%%%%%%%%
% Stores the 3 lines forming a triangle
tris = zeros(not1+not2, 3);

% Read in triangles from 1st GTS
tris = t1;
t_count = not1+1; % next available slot

% Read in triangles from 2nd GTS
for n=1:not2,
    tmp = t2(n,:);

    % Map into join reference frame
    entry = lmap([t2(n,:)],:)';

    index = exists(tris, entry);

    if index == 0,
        tris(t_count,:) = entry;
        t_count=t_count+1;
    end
end
```

Function : gtsmergeall

```
function [] = gtsmergeall(ofname, prefix, suffix, nsegments)
% gtsmergeall(ofname,prefix,suffix, nsegments)
%
% Uses gtsmergefiles to merge all segments produced by MEPtri.m
discretisation
% method.
%
% ofname - string, filename of final surface
% prefix - string, common filename prefix of all segments
% suffix - string, common filename suffix of all segments, e.g '.gts'
% nsegments - total number of segments
%
% Sujee Mampitiyarachchi, 9/07/06
```

```
%suffix = '.gts';

%all = strcat(prefix,suffix);

for i=1:nsegments,
    fname=strcat(strcat(prefix, num2str(i)), suffix);
    if i==2,
        gtsmergefiles(ofname, strcat(strcat(prefix, num2str(i-1)),
suffix), fname);
    end
    if i>2,
        gtsmergefiles(ofname, ofname, fname);
    end
end
```

Function : gtsmergefiles

```
function [] = gtsmergefiles(ofname, fname1, fname2)
% gtsmergefiles(ofname, fname1, fname2)
%
% Join 2 GTS files named fname1 and fname2
% Does not recognise intersecting faces.
%
% ofname being the name of the output file containing
% the GTS representation of the merged surface.
%
% Sujee Mampitiyarachchi, 8/07/06

[p1, l1, t1] = gtsload(fname1);
[p2, l2, t2] = gtsload(fname2);

[points,lines,tri]=gtsmerge(p1, l1, t1, p2, l2, t2);

gtssave(ofname, points, lines, tri);
```

Viewing Functions

Function : gtsview

```
function [] = gtsview(points, lines, tris)
% gtsview(points, lines, tris)
%
% Plot GTS surface represented by points, lines, tris
%
% Sujee Mampitiyarachchi 14/07/06
figure(1)
hold on;
px = zeros(3,1);
py = zeros(3,1);
pz = zeros(3,1);

TWOD = 0;
```

```
b = size(points);
if b(2) == 2,
    TWOD = 1;
end

for m=1:length(tris),

    lA=tris(m,1);
    lB=tris(m,2);
    lC=tris(m,3);

    px=points(lines(lA,[1 2]),1);
    py=points(lines(lA,[1 2]),2);
    pz=zeros(length(py),1);

    if ~TWOD,
        pz=points(lines(lA,[1 2]),3);
    end

    plot3(pz,px,py,'-b');

    px=points(lines(lB,[1 2]),1);
    py=points(lines(lB,[1 2]),2);
    pz=zeros(length(py),1);

    if ~TWOD,
        pz=points(lines(lB,[1 2]),3);
    end

    plot3(pz,px,py,'-b');

    px=points(lines(lC,[1 2]),1);
    py=points(lines(lC,[1 2]),2);
    pz=zeros(length(py),1);

    if ~TWOD,
        pz=points(lines(lC,[1 2]),3);
    end

    plot3(pz,px,py,'-b');

end

view(3)
legend off
grid off
axis equal
```

Function : gtsviewfile

```
function [] = gtsviewfile(fname)
% gtsviewfile(fname)
%
% Plot GTS surface represented by fname
%
```

```
% fname     -   name of file to read surface from
%               Must be in gts format.
%
% Sujee Mampitiyarachchi 29/03/06

% Load GTS data from file
[points, lines, tris] = gtsload(fname);

gtsview(points,lines,tris);
```

Airfoil Functions

Functon : naca4series

```
function [ad] = naca4series(m, p, t, nx)
% ad = naca4series(m, p, t,nx)
%
% Generates an airfoil using the naca 4 series analytical
% equations. Defining the trailing edge as the point where
% thickness is zero.
%
% m,p and t must be in terms of chord.
% m is the maximum camber
% p is the location of maximum camber
% t is the maximum thickness of airfoil
%
% e.g for NACA2415 = naca4series(0.02,0.4,0.15, 50);
%
% returns a matrix ad containing the [x,y] datapoints.
%
% The number of datapoints returned is equal to 2*nx+1
%
% Sujee Mampitiyarachchi, 19/9/06

xte = fsolve(@nacathickness,1);
xle = 0;

% Leading Edge Radius
r=1.10*t*t;

if p == 0,
    p = 0.5;
end

if (p~=0 && m~=0),

    xfp=[0:2/nx:1]';
    % Distribute more stations at the leading edge
    xfp=p*(1-cos(pi/2*xfp)+xle/p);

    yfp=m/p/p*(2*p.*xfp-xfp.*xfp);
    dydxfp = m/p/p*(2*p-2.*xfp);
```

```
    xap=[0:2/nx:1]';
    % Distribute more stations to the trailing edge
    xap=(xte-p-(xte-xle)/nx)*sin(pi/2*xap)+p+(xte-xle)/nx;

    yap = m/(1-p)/(1-p)*((1-2*p)+2*p.*xap-xap.*xap);
    dydxap = m/(1-p)/(1-p)*(2*p-2*xap);

    dydx=[dydxfp;
        dydxap];

    theta = atan(dydx);

    x=[xfp;
       xap];

    yc=[yfp;
        yap];

else
    x = [0:xte/nx:xte]'

    %x=xte-cos(pi/2/xte*x);
    x=sin(pi/2*[-1:2/nx:1]');
    x=xte/2*(x+1);

    yc = zeros(length(x),1);
    theta = zeros(length(x),1);
end

yt=t/0.20*nacathickness(x);

xu = x - yt.*sin(theta);
%[x,xu]
yu = yc + yt.*cos(theta);

xl = x + yt.*sin(theta);

yl = yc - yt.*cos(theta);

% Origin of Leading Edge Radius
if(p~=0 && m~=0)

xo = noncomplx(roots([1,-4*p,4*p*p+p^4/m/m,0,-p^4*r*r/m/m]));

yo = m/p/p*(2*p*xo-xo*xo);

else
xo = -r;
yo = 0;
end

for i=1:length(xl),
    if xl(i)<xo*1e-4,
        yl(i)=-1*sqrt(r*r-(xl(i)-xo)*(xl(i)-xo))+yo;
  %      xl(i)
    end
end
```

```
for c=1:length(xu),
    if xu(c)<xo*1e-4,
        yu(c)=sqrt(r*r-(xu(c)-xo)*(xu(c)-xo))+yo;

    end
end

au = sortSafe([xu,yu],1, 1);
ax=[au([2:length(au)],1);
    xl([2:length(xl)])];
ay=[au([2:length(au)],2);
    yl([2:length(xl)])];

ad=[ax,ay];
```

Function : nacathickness

```
function [y] = nacathickness(x)
% [y] = nacathickness(x)
%
% Evaluates the thickness given by equation
% for thickness used in the naca 4 and 5 series
% airfoils.
%
% Sujee Mampitiyarachchi, 26/8/06

y=0.29690*x.^(1/2)-0.12600*x-0.35160*x.*x+0.28430*x.*x.*x-0.10150*x.^4;
```

Function : noncomplex

```
function [res] = noncomplx(root)
% [res] = noncomplx(roots)
%
% Returns the first real root.
%
% Sujee Mampitiyarachchi, 26/8/06

c = length(root);

for i=1:c,
    if isreal(root(i)) && root(i)>0,
       res = root(i);
       break;
    end
end
```

Other

Function : gtsload

```
function [points, lines, tris] = gtsload(fname)
```

```
% [points, lines, tri] = gtsload(fname)
%
% Loads data from a GTS file specified by fname
%
% Sujee Mampitiyarachchi, 12/07/06

if (ischar(fname)~=true),
   disp('**ERROR** Incompatible input arguement - fname not a char
array')
   disp('Exiting function')
end

fid = fopen(fname,'r');

if fid==-1,
    disp(sprintf('**ERROR** unable to open %s', fname))
end

% Read header string
header = fgetl(fid);

% Number of Points
[T,R]=strtok(header);
NoP=str2num(T);

% Number of Lines
[T,R]=strtok(R);
NoL=str2num(T);

% Number of Triangles
[T,R]=strtok(R);
NoT=str2num(T);

% GtsObject representing Triangles
[tObj,R]=strtok(R);

% GtsObject representing Lines
[lObj,R]=strtok(R);

% GtsObject representing Points
[pObj,R]=strtok(R);

points = zeros(NoP,3);
lines = zeros(NoL,2);
tris = zeros(NoT,3);

% Read in the points
for i=1:NoP,
    points(i,:)=sscanf(fgetl(fid),'%f').';
end

% Construct the lines
for j=1:NoL,
    lines(j,:)=sscanf(fgetl(fid), '%d').';
end

% Collate the triangles
```

```
for k=1:NoT,
    tris(k,:)=sscanf(fgetl(fid), '%d').';
end
```

Function : gtssave

```
function [] = gtssave(fname, point, line, tri)
% function [] = gtssave(fname, point, line, tri)
%
% Writes a GTS representation to a file specified by fname
% Using the GtsSurface, GtsFace, GtsEdge and GtsVertex
% objects to represent in Gerris.
%
% Sujee Mampitiyarachchi, 13/7/06

% Print vertices, lines, triangles in GTS format.
out = fopen(fname,'w');

fprintf(out, '%d %d %d GtsSurface GtsFace GtsEdge GtsVertex\n',
length(point), length(line), length(tri));

fprintf(out, '%f %f %f\n', point');

fprintf(out, '%d %d\n', line');

for i=1:length(tri),
    if ~(tri(i,1)==0 && tri(i,2)==0 && tri(i,3)==0),
        fprintf(out, '%d %d %d\n', tri(i,1), tri(i,2), tri(i,3));
    end
end

fclose(out);
```

Function : gtspeek

```
function [] = gtspeek(fname)
%[] = gtspeek(fname)
%
% Prints a description of the GTS surface specified by fname
%
% fname - filename of a gts surface
%
% Sujee Mampitiyarachchi, 19/9/06

[p,l,t]=gtsload(fname);

display(['Number of Vertices  :',int2str(length(p))])
display(['Number of Edges     :', int2str(length(l))])
display(['Number of Triangles :', int2str(length(t))])

display(['Min X: ', num2str(min(p(:,1))), ' Max X: ',
num2str(max(p(:,1)))])
display(['Min Y: ', num2str(min(p(:,2))), ' Max Y: ',
num2str(max(p(:,2)))])
```

```
display(['Min Z: ', num2str(min(p(:,3))), ' Max Z: ',
num2str(max(p(:,3)))])
```

Function : sortSafe

```
function [sorted]=sortSafe(data, col, direction)
% [sorted]=sortSafe(data, col, direction)
%
% Sorts an array by specified col while maintaining
% the data in each row.
%
% direction : 1 for descending
%             0 for ascending
%
% Sujee Mampitiyarachchi, 18/07/06

if direction == 1,
[Y,I]=sort(data, col, 'descend');
else
    [Y,I]=sort(data, col, 'ascend');
end

for i=1:length(data),
    sorted(i,:)=data(I(i,1),:);
end
```

Function : wpdata

```
function [plane1, plane2, z1, z2] = wpdata(points, lines, tris)
% [plane1, plane2, z1, z2] = wpdata(points, lines, tris)
%
% Winglet Plane Data
%
% Determines winglet plane of GTS surface represented by points, lines,
% tris. The extreme min and max Z value are the locations of the
winglet
% planes. plane1 and plane2 are the profile section data at z1 and z2
% respectively.
%
% Sujee Mampitiyarachchi, 14/7/06

% Determine z-value of the wingtip planes of surface
z2 = max(points(:,3));
z1 = min(points(:,3));

count=0;

% Determine the section profile of the wingtip planes
for i=1:length(points),
    if points(i,3) == z1,
        plane1(i,:)=points(i,[1 2]);
    end
    if points(i,3) == z2,
        count=count+1;
        plane2(count,:)=points(i,[1 2]);
```

```
    end
end
```

Function : gtsRotate

```
function [newprofile]=gtsRotate(profile,theta, ax, ref)
% [newprofile]=gtsRotate(profile,theta, ax, ref)
%
% Rotate airfoil sections
% profile - matrix of 3 columns, [x,y,z] defining airfoil sections
% theta - rotation angle in radians
% ax - 'x','y' or 'z' axis.
% ref - [x,y,z] reference point i.e origin
%
% Sujee Mampitiyarachchi, 3/8/06

Rz=[cos(theta) -sin(theta) 0;
    sin(theta) cos(theta) 0;
    0 0 1];

Ry=[cos(theta), 0, sin(theta);
    0, 1, 0;
    -sin(theta), 0, cos(theta)];

Rx=[1 0 0;
    0 cos(theta) -sin(theta);
    0 sin(theta) cos(theta)];

if ax=='z',
newprofile=(profile-
ones(length(profile),1)*ref)*Rz+ones(length(profile),1)*ref;
else if ax=='y',
        newprofile=(profile-
ones(length(profile),1)*ref)*Ry+ones(length(profile),1)*ref;
    else if ax=='x',
            newprofile=(profile-
ones(length(profile),1)*ref)*Rx+ones(length(profile),1)*ref;
        end
    end
end

newprofile=round(1e6*newprofile)/1e6;
```

Function : neaten

Appendix B - Gerris Commands Reference.

The official reference manual for the Gerris Flow Solver can be found at **gfs.sourceforge.net** , however this appendix contains alternate descriptions of commands and useful ways to use them.

GfsEventScript

Syntax: GfsEventScript {} {}

Used to execute shell commands from with in a simulation.

Examples:

```
GfsEventScript {istep = 10} {
echo –n $GfsTime >> memloading.txt
free | gawk '/Swap/ {printf " %d\n, $4}' >> memloading.txt
}
```

GfsRefineSolid

Syntax: `GfsSolidRefine  x`

Define a refinement level on cells cut by the solid boundary.

Examples:

```
GfsSolidRefine {
      if(SolidCurvature < 0.){
            gint l = log(SolidCurvature/0.2)/log(2.)
            return l < 8 ? 8 : l > 12 ? 12 : l;
      }
      else
            return 8;
}
```

GfsSolidForce

Syntax: `GfsSolidForce FILE`

GfsOutputLocation

Syntax: `GfsOutputLocation FILE1 FILE2`

Where FILE1 is the name of the output file. FILE2 contains the list of points to evaluate in [x y z] form.

Examples:

To obtain the pressure/velocity distributions about an airfoil, use the GfsOutputLocation command. In FILE2 list the points that produce the airfoil profile.

Appendix C - Simulation Data

This section contains the complete Gerris scripts used to run the simulations in the grid convergence study and the main three-dimensional study.

Ref0

```
3 2 GfsSimulation GfsBox GfsGEdge {} {

      GfsTime {end = 10}

      GfsRefine  4

      GfsRefineSolid {
            if(SolidCurvature > 0.){
            gdouble l = log(SolidCurvature/0.2)/log(2.);
            return l < 8 ? 8 : l > 10 ? 10 : l;
            }else
                  return 8;
      }

      GfsInit {} {U=1 V=0}

      # 2D Modelling of NACA0012 airfoil
      GtsSurfaceFile iorig.gts

      # Output Events
      GfsOutputTime            {istep = 10} time.txt
      GfsOutputSolidForce {step = 0.01} forces.txt

      # Saving simulation to file
      GfsOutputSimulation { step = 0.1 } sec-%1.1f.gfs{}

      # At the end of sim, append timing summary
      GfsOutputTiming { start = end } time.txt
```

```
      GfsOutputLocation { start=end } upper.prof upper.0012
      GfsOutputLocation { start=end } lower.prof lower.0012

      GfsOutputBalance {start=end } stdout
}
GfsBox{ left = GfsBoundaryInflowConstant 1}
GfsBox{}
GfsBox{right = GfsBoundaryOutflow}
1 2 right
2 3 right
-------------------------------------------------------------
```

Ref1

```
3 2 GfsSimulation GfsBox GfsGEdge {} {

      GfsTime {end = 10}

      GfsRefine  4

      GfsRefineSolid {
            if(SolidCurvature > 0.){
            gdouble l = log(SolidCurvature/0.2)/log(2.);
            return l < 9 ? 9 : l > 11 ? 11 : l;
            }else
                  return 9;
      }

      GfsInit {} {U=1 V=0}

      GfsOutputTime            {istep = 10} time.txt
      GfsOutputSolidForce {step = 0.01} forces.txt

      # Saving simulation to file
      GfsOutputSimulation { step = 1 } sec-%1f.gfs{}

      # At the end of sim, append timing summary
      GfsOutputTiming { start = end } time.txt

      GfsOutputLocation { start=end } upper.prof upper.0012
      GfsOutputLocation { start=end } lower.prof lower.0012

      GfsOutputBalance {start=end } stdout
}
GfsBox{ left = GfsBoundaryInflowConstant 1}
GfsBox{}
GfsBox{right = GfsBoundaryOutflow}
1 2 right
2 3 right

-------------------------------------------------------------
```

Ref2

```
3 2 GfsSimulation GfsBox GfsGEdge {} {

      GfsTime {end = 10}

      GfsRefine  4

      # Apply a fine mesh at leading edge (<5% chord
      GfsRefineSolid {
            return x<0.025 ? 12 : x>0.195 ? 12 : 11;
      }

      GfsInit {} {U=1 V=0}

      # 2D Modelling of NACA0012 airfoil
      GtsSurfaceFile imy.gts

      # Output Events
      GfsOutputTime            {istep = 10} time.txt
      GfsOutputSolidForce {step = 0.01} forces.txt

      # Saving simulation to file
      GfsOutputSimulation { step = 1 } sec-%1f.gfs{}

      # At the end of sim, append timing summary
      GfsOutputTiming { start = end } time.txt

      GfsOutputLocation { start=end } upper.prof upper.0012
      GfsOutputLocation { start=end } lower.prof lower.0012

      GfsOutputBalance {start=end } stdout

}
GfsBox{ left = GfsBoundaryInflowConstant 1}
GfsBox{}
GfsBox{right = GfsBoundaryOutflow}
1 2 right
2 3 right
```

Ref3

```
3 2 GfsSimulation GfsBox GfsGEdge {} {

      GfsTime {end = 10}

      GfsRefine  4

      # Apply a fine mesh at leading edge (<5% chord
      GfsRefineSolid {
            return x<0.025 ? 10 : x>0.195 ? 14 : 9 ;
      }

      GfsInit {} {U=1 V=0}

      # 2D Modelling of NACA0012 airfoil
```

```
      GtsSurfaceFile imy.gts

      GfsOutputTime            {istep = 10} time.txt
      GfsOutputSolidForce {step = 0.01} forces.txt

      # Saving simulation to file
      GfsOutputSimulation { step = 1 } sec-%1f.gfs{}

      # At the end of sim, append timing summary
      GfsOutputTiming { start = end } time.txt



      GfsOutputLocation { start=end } upper.prof upper.0012
      GfsOutputLocation { start=end } lower.prof lower.0012

      GfsOutputBalance {start=end } stdout


}
GfsBox{ left = GfsBoundaryInflowConstant 1}
GfsBox{}
GfsBox{right = GfsBoundaryOutflow}
1 2 right
2 3 right
```

Ref4

```
3 2 GfsSimulation GfsBox GfsGEdge {} {

      GfsTime {end = 10}

      GfsRefine  4

      # Apply a fine mesh at leading edge (<5% chord
      GfsRefineSolid {
                if(SolidCurvature > 0.){
                        gint l = log(SolidCurvature/0.2)/log(2.);
                        return l < 8 ? 8 : l > 10 ? 10 : l;
                }
                else
                        return 8;
        }

      GfsInit {} {U=1 V=0}

      # 2D Modelling of NACA2412 airfoil
      GtsSurfaceFile wing.gts

      # Output Events
      GfsOutputTime            {istep = 10} time.txt
      GfsOutputSolidForce {step = 0.01} forces.txt

      # Saving simulation to file
```

```
      GfsOutputSimulation { step = 0.1 } sec-%1.1f.gfs{}

      # At the end of sim, append timing summary
      GfsOutputTiming { start = end } time.txt

      GfsOutputLocation { start=end } upper.prof upper.2412
      GfsOutputLocation { start=end } lower.prof lower.2412

      GfsOutputBalance {start=end } stdout
}
GfsBox{ left = GfsBoundaryInflowConstant 1}
GfsBox{}
GfsBox{right = GfsBoundaryOutflow}
1 2 right
2 3 right
```

Ref5

```
3 2 GfsSimulation GfsBox GfsGEdge {} {

      GfsTime {end = 10}

      GfsRefine  4

      # Apply a fine mesh at leading edge (<5% chord
      GfsRefineSolid 10

      GfsInit {} {U=1 V=0}

      # 2D Modelling of NACA2412 airfoil
      GtsSurfaceFile wing.gts

      GfsAdaptVorticity {istep=1} {maxlevel = 8 cmax=1e-2}

      # Output Events
      GfsOutputTime            {istep = 10} time.txt
      GfsOutputSolidForce {step = 0.01} forces.txt
      # Saving simulation to file
      GfsOutputSimulation { step = 0.1 } sec-%1.1f.gfs{}

      # At the end of sim, append timing summary
      GfsOutputTiming { start = end } time.txt

      GfsOutputLocation { start=end } upper.prof upper.2412
      GfsOutputLocation { start=end } lower.prof lower.2412

      GfsOutputBalance {start=end } stdout
}
GfsBox{ left = GfsBoundaryInflowConstant 1}
GfsBox{}
GfsBox{right = GfsBoundaryOutflow}
1 2 right
2 3 right
```

Ref6

```
3 2 GfsSimulation GfsBox GfsGEdge {} {

      GfsTime {end = 10}

      GfsRefine  4

      # Apply a fine mesh at leading edge (<5% chord
      GfsRefineSolid 10

      GfsInit {} {U=1 V=0}

      # 2D Modelling of NACA2412 airfoil
      GtsSurfaceFile wing.gts

      GfsAdaptVorticity {istep=1} {maxlevel = 8 cmax=1e-2}

      GfsOutputTime            {istep = 10} time.txt
      GfsOutputSolidForce {step = 0.01} forces.txt

      # Saving simulation to file
      GfsOutputSimulation { step = 1 } sec-%1.1f.gfs{}

      # At the end of sim, append timing summary
      GfsOutputTiming { start = end } time.txt

      GfsOutputLocation { start=end } upper.prof upper.2412
      GfsOutputLocation { start=end } lower.prof lower.2412

      GfsOutputBalance {start=end } stdout

}
GfsBox{ left = GfsBoundaryInflowConstant 1}
GfsBox{}
GfsBox{right = GfsBoundaryOutflow}
1 2 right
2 3 right
```

Ref7

```
3 2 GfsSimulation GfsBox GfsGEdge {} {

      GfsTime {end = 10}

      GfsRefine  4

      # Apply a fine mesh at leading edge (<5% chord
      GfsRefineSolid {
```

```
                if(SolidCurvature > 0.){
                        gint l = log(SolidCurvature/0.2)/log(2.);
                        return l < 8 ? 8 : l > 10 ? 10 : l;
                }
                else
                        return 8;
          }

        GfsInit {} {U=1 V=0}

        # 2D Modelling of NACA2412 airfoil
        GtsSurfaceFile wing.gts

        GfsAdaptVorticity {istep=1} {cmax=1e-2 maxlevel=8}

        GfsOutputBalance  {istep = 10} domain.txt
        GfsOutputTime            {istep = 10} time.txt
        GfsOutputSolidForce {step = 0.01} forces.txt

        # Saving simulation to file
        GfsOutputSimulation { step = 0.1 } sec-%1.1f.gfs{}

        # At the end of sim, append timing summary
        GfsOutputTiming { start = end } time.txt

        GfsOutputLocation { start=end } upper.prof upper.2412
        GfsOutputLocation { start=end } lower.prof lower.2412

}
GfsBox{ left = GfsBoundaryInflowConstant 1}
GfsBox{}
GfsBox{right = GfsBoundaryOutflow}
1 2 right
2 3 right

-------------------------------------------------------------
```

wB

```
3 2 GfsSimulation GfsBox GfsGEdge {} {

        GfsTime {end = 10}

        GfsRefine 4

        # Apply a fine mesh at leading edge (<5% chord)
          GfsRefineSolid {
                if(SolidCurvature > 0.){
                        gint l = log(SolidCurvature/0.2)/log(2.);
                        return l < 9 ? 9 : l > 12 ? 12 : l;
                }
                else
                        return 9;
          }

        # Initial Condition
```

```
        GfsInit {} {U=1 V=0}

        # Basic Wing as specified in Jarno Montinnen's Thesis
        # E212 airfoil, in a rectangular planform wing with
        # 17 deg l.e sweep, AR=1, TIP_CHORD = 0.8467
        # with winglet B, 65 deg l.e sweep, x_offset = 0.25, span = 0.12,
        # taper_ratio = 0.66
        GtsSurfaceFile wing_wb.gts

        # Adaptation
        GfsAdaptVorticity {istep=1} { maxlevel =5 cmax= 1e-2 }

        # Every iteration output the GfsTime Swap Memory Free
        GfsEventScript    {istep = 1} {
        echo -n $GfsTime >> memloading.txt
        free | gawk '/Swap/ {printf " %d\n", $4}' >> memloading.txt
        }

        # Output Events
        GfsOutputBalance  {istep = 10} domain.txt
        GfsOutputTime            {istep = 10} time.txt
        #GfsOutputProjectionStats { istep = 10 } proj.txt
        GfsOutputSolidForce {step = 0.01} forces.txt

        # Saving simulation to file
        GfsOutputSimulation { step = 0.1 } qua-%1.1f.gfs{}

        # At the end of sim, append timing summary
        GfsOutputTiming { start = end } time.txt


        GfsOutputLocation {istep=8000} c20.prof c20.e212
        GfsOutputLocation {istep=8000} c40.prof c40.e212
        GfsOutputLocation {istep=8000} c60.prof c60.e212
        GfsOutputLocation {istep=8000} c80.prof c80.e212
        GfsOutputLocation {istep=8000} c98.prof c98.e212

        # At the end of the sim, filter the domain.txt to
        # obtain a list of the average domain size.
        GfsEventScript { start = end } {
        cat domain.txt | gawk '/avg:/ {print $4}' > domain
        rm -f domain.txt
        }
}
GfsBox{ left = GfsBoundaryInflowConstant 1}
GfsBox{}
GfsBox{right = GfsBoundaryOutflow}
1 2 right
2 3 right
```