12-2017

Development of a Multi-Jointed Wing Surface Mover

Collin Arthur Strassburger
University of Tennessee, Knoxville, cstrassb@vols.utk.edu

To the Graduate Council:

I am submitting herewith a thesis written by Collin Arthur Strassburger entitled "Development of a Multi-Jointed Wing Surface Mover." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Aerospace Engineering.

Kivanc Ekici, Major Professor

We have read this thesis and recommend its acceptance:

James Coder, Zhili Zhang, Jay I. Frankel

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Development of a Multi-Jointed Wing Surface Mover

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Collin Arthur Strassburger

December 2017

For those who have made me who I am today.

Acknowledgments

I would like to thank my parents for providing their support; I could never have done this without them.

I would also like to thank my advisor and my fellow graduate students for all their advise and encouragement.

The great bird will take its first flight . . . filling the world with amazement and all records with its fame, and it will bring eternal glory to the nest where it was born.
Leonardo da Vinci

Abstract

The field of ornithopter research has reached a point where it has become commonplace for Computational Fluid Dynamics (CFD) solvers to have built-in capabilities for rigid solid body motion. This is suitable for micro air vehicles (MAVs) yet is often not flexible enough to model wings with dynamic internal structure, such as the wings of birds and bats. There is currently no program available to perform the surface motion of a wing which has multiple independently moving joints. The code, detailed in this paper, provides the user with this type of capability. The bone lengths, joint angle properties, and thickening parameters are input and the progressive motion of surface points for each desired time is output. Furthermore, an optimized minimal surface solver is included for use with elastic wings. The output of this code has been integrated with OpenFOAM to provide proof-of-concept and verification results. The verification results demonstrate that both the process and code are viable while the 3D surface motion results demonstrate the motion of a pterosaur wing. As a result, this code opens the door to a large region of unexplored behaviors and properties which stem from highly dynamic multi-jointed wings.

Table of Contents

List of Tables

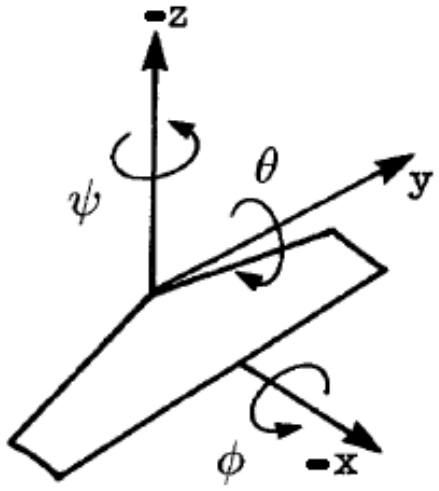
List of Figures

Nomenclature

$\eta(x, y)$	Error between the approximate and correct z-coordinate of the point at coordinates (x,y)
ν	Kinematic viscosity
ω	Reduced Frequency
Φ	Dihedral angle
Φ_i	Dihedral angle of joint i
Φ_{mod}	Dihedral angle modification
Ψ	Sweep angle
Ψ_i	Sweep angle of joint i
Ψ_{mod}	Sweep angle modification
ρ	Density
ρ_∞	Freestream density
Θ	Pitch angle
θ	Rotational angle of airfoil at c/4
θ_0	Amplitude of airfoil rotation at c/4

Θ_i	Pitch angle of joint i
φ	Phase shift between rotational and height parameters for airfoil motion
φ_{A_i}	Phase shift of motion component A_i
$A_{i,actual}$	Angular component A of joint i
$A_{i,glide}$	Angular component A of joint i in the glide configuration
A_i	Angular componen of joint i to be replaced with Ψ, Θ, or Φ
c	Airfoil chord length
C_L	Lift coefficient
C_p	Pressure coefficient
f	Reduced frequency of airfoil motion
h	Airfoil height at c/4
h_0	Amplitude of airfoil height at c/4
k	Iteration number
L	Lift generated by the wing
P	Pressure
S_{ref}	Wing reference area
t	Time
u	Velocity
u_∞	Freestream velocity

$Z(x, y)$	Approximate z-coordinate of the point at coordinates (x,y)
$z(x, y)$	Correct z-coordinate of the point at coordinates (x,y)
AoA	Angle of Attack
CFD	Conputational fluid dynamics
DoF	Degrees of freedom
FSI	Fluid structure interaction
fvMotion	Finite volume mesh motion
ILES	Implicit large eddy simulation
k	Airfoil reduced frequency
LES	Large eddy simulation
MAV	Micro air vehicle
PDE	Partical differenetial equation
PIMPLE	Merged PISO-Simple algorithm
PISO	Pressure implicit with splitting of operator
PIV	Particle image velocimetry
RAS	Reynolds averaged simulation
RBF	Radial basis function
Re	Reynolds number
SA	Spalart-Allmaras turbulence model
t/T	Non-dimensionalized time as fraction of period



Axes, modified from Sträng (2009)

Chapter 1

Introduction

Numerical models for fixed-wing aircraft flight are present in abundance as the mechanics are well understood and relatively intuitive. For flapping flight, there has been a great deal of work to develop models for micro air vehicles due to the potential military and search and rescue applications.

Analysis of bird flight began prior to the mid-1980s. In a paper by Spedding (1987), the researchers were analyzing the wake of *Falco Tinnunculus* (a type of Kestrel) to validate the elliptically loaded airfoil as the loading distribution used during biological gliding flight. As time progressed, the imaging equipment became more capable allowing Spedding et al. (2003) to use laser based PIV rather than the helium filled bubbles used previously. These technological enhancements allowed for increasing amounts of data, which correspondingly allowed for more detailed analysis. In a paper by Spedding et al. (2003), the analysis is focused on determining the forces and power requirements of the freely flying bird, rather than the steady-state behavior analyzed previously. Hall et al. (1998) presented the first computational model for large-amplitude flapping flight which utilized a vortex-lattice model of the wake.

Once computational power became sufficiently economical, the aerodynamics of insect flight came under scrutiny. Gopalakrishnan (2008) performed an analysis of these creatures to determine the effects of Reynolds numbers (Re) upon their

performance. In a similar effort, Jones (2013) performed an analysis with the goal of optimizing a Micro Air Vehicle's (MAV) wing kinematics and structure for flying in a gusty environment. In conjunction with these analyses, there have been CFD packages specifically developed for low-Re flapping flight. One such example is detailed in a paper by Pearson et al. (2011) which utilizes a variety of techniques suitable for low-Re flight.

Despite the abundance of information concerning MAVs, there is a distinct lack of quantitative understanding regarding the flight mechanics of larger ornithopters, such as pterosaurs, whose flight regimes yield significantly higher Reynolds numbers than their micro-scale cousins. There are many qualitative papers which postulate about the flight capabilities of these creatures by analyzing their wing characteristics and bones, such as those by de Ricqlès et al. (2000), Wilkinson (2008), and Witton and Habib (2010).

The optimal flap patterns for one particular pterosaur, *Coloborhynchus robustus*, have been calculated by Sträng (2009). Sträng utilized low fidelity aerodynamic models in conjunction with multi-step optimization techniques to determine the flap patterns given a variety of different conditions. However, there was no way to determine the validity of the low fidelity models for this flight regime.

To test the low fidelity aerodynamic model used by Sträng (2009), a high fidelity aerodynamic model must be used. These high fidelity models similarly require a high resolution model for the motion of the wing. However, at present, no high resolution motion models have the capabilities required for multi-jointed wings; namely, the ability to have multiple motion joints acting simultaneously through time.

When a joint is not at the end of a body, that body must deform to accommodate the joint. For simple joints, this will often result in one body passing through the other. Another implementation is to have the body around the joints deform to accommodate the motion; for example, when a human finger is bent, the skin on the lengthening side stretches and the skin on the contracting side creases. However, this is not a viable implementation for motion models as it is very easy to generate

inverted cells at the crease. As such, the body must smoothly deform around the joint; much like when the arm is pivoted in the shoulder. When the body undergoes this smooth deformation, the distance to the nearest bone is maintained yet no crease forms; it is this type of joint-body interaction which must be modeled in the motion code.

Elastic wings, such as those employed by bats, create another issue: they form a minimal surface. This is caused by the elastic tension inherent to the membrane and results in the equilibrium state where there is the minimum amount of area connecting the specified boundary conditions. As this is the type of wing modeled by Sträng (2009), the wing motion code must be able to accommodate an elastic wing. To do this, the motion code must include a partial differential equation (PDE) solver for the elliptical minimal surface equation. This type of solver utilizes finite difference approximations and an iterative approach to determine the solution to the PDE.

The code presented and validated in this paper meets these criterion and allows the user to move a wing with up to 7 joints, where each joint has 3 degrees of freedom, in any arbitrary periodic pattern with minor constraints. The code outputs the initial surface for a meshing program, moves the surface for dynamic cases, and allows for minimal surface wing structures.

Chapter 2

Setup

2.1 Wing Information

The pterosaur of study, *Coloborhynchus robustus*, has known bone lengths and estimated positions as used by Sträng (2009). The skeletal structure is shown in Figure 2.1, while the joint positions are given in Table 2.1. The glide joint angles are shown in Table 2.2

It was found that combining these joint angles with the specified joint locations does not yield the planar glide plot given. As the author and principal advisor were unavailable, alternative methods to attain the given glide plots were required. It is suspected that greater accuracy of these values must be attained, as rounding to the nearest degree could certainly result in unacceptable levels of inaccuracy. It is assumed that each joint is off by the same magnitude; as such only 2 modifiers need to be obtained, Φ_{mod}, and Ψ_{mod} which correspond to the dihedral angle modification and the sweep angle modification, respectively. Upon inspecting the geometry plot (Figure 2.2), it is apparent that the 4th joint is foremost in the flight direction. Furthermore, upon measuring of the joint positions in multiple manners, the targets and relative accuracies are determined for the 4th and 8th joints, see Table 2.3, where the 8th joint is the wing tip and the 1st joint is the shoulder.

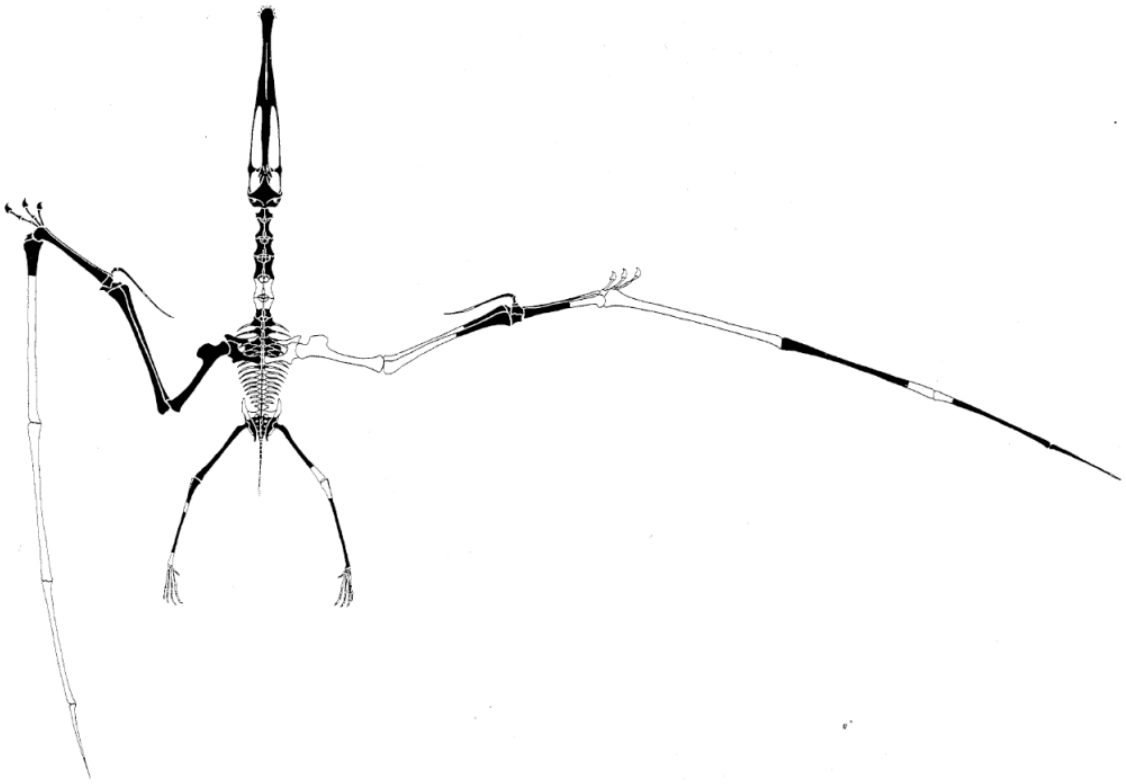


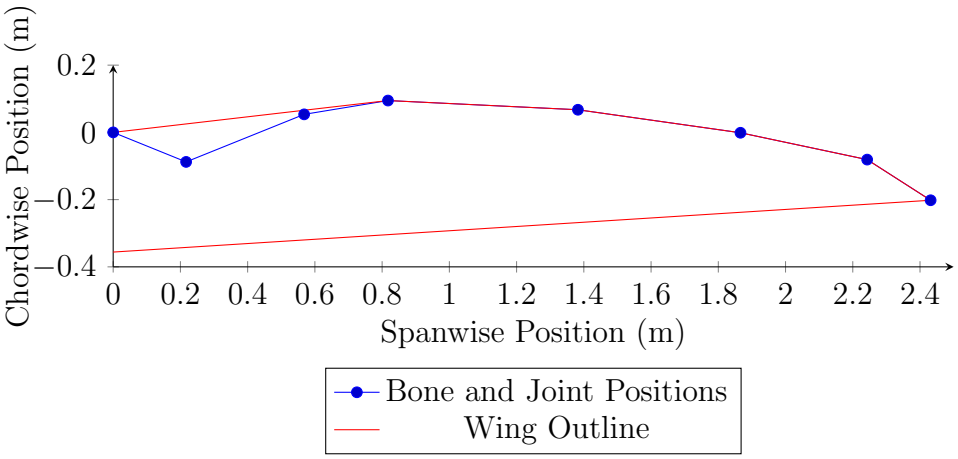
Figure 2.1: Outline of skeleton of *Coloborhynchus robustus* with the left wing folded and the right wing in the flying position. Black indicates the preserved bones on specimen NSM-PV 19892. Note that this skeleton was originally classified as *Anhanguera piscator*. Kellner and Tomida (2000) and Sträng (2009)

Table 2.1: Joint positions and corresponding chord lengths of *Coloborhynchus robustus*. Sträng (2009)

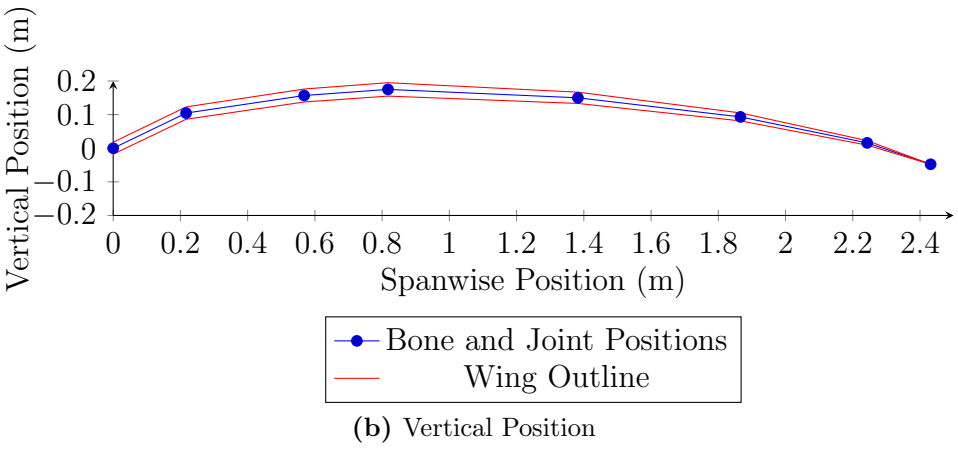
Joint Name	Spanwise Position	Chord
	mm	mm
Shoulder	0	356
Elbow	240	351
Wrist	599	343
Knuckle	852	328
Phalange I-II	1413	295
Phalange II-III	1899	216
Phalange III-IV	2277	120
Wing Tip	2485	0

Table 2.2: Glide Joint Angles for *Coloborhynchus robustus*. Sträng (2009) Ph. stands for interphalangeal joint. All values are in degrees.

Joint Name	Shoulder	Elbow	Wrist	Knuckle	Ph.I-II	Ph.II-III	Ph.III-IV
Joint Index	1	2	3	4	5	6	7
Φ_i	23	-9	-6	-10	-5	-6	-8
Θ_i	0	0	0	0	0	0	0
Ψ_i	-22	43	-10	-10	-5	-2	-18



(a) Chordwise Location



(b) Vertical Position

Figure 2.2: Joint positions during gliding flight. Adapted from Sträng (2009)

Table 2.3: Extrapolated joint position data.

	Joint 4			Joint 8		
	X	Y	Z	X	Y	Z
Target	-0.20832	2.42619	0.054	0.0912	0.809524	-0.1752
Relative Accuracy	2.20625	10.6995	1.536458	0.90625	3.57	0.541667

Table 2.4: Variable modifications

Variable	Degrees
Ψ_{mod}	-0.56
Φ_{mod}	-0.07

By performing a 0^{th} order optimization, i.e. selecting the best result of those tested, with a resolution of 0.01 degrees and sorting the results by least squares of the difference multiplied by the relative accuracy for each component of each joint location, the requisite variable modifications were obtained and are shown in Table 2.4. The 1 million combinations were evaluated and those evaluations took approximately 1 hour of computing time on a quad-core machine.

These results are applied to all of the corresponding angles and yield the optimal match for the plot given by Sträng (2009). For the actuation case, the joint motion is specified by Eqn 2.1 wherein A is replaced with Φ, Ψ, or θ, as defined in the nomenclature, while i is substituted with the joint index and φ indicates the phase angle. See Table 2.5 for the substitution values.

$$A_{i,actual} = A_{i,glide} + A_i * sin(\omega * t + \varphi_{A_i}) \tag{2.1}$$

Table 2.5: Joint Index Conversion

Parameter	Index
Shoulder	1
Elbow	2
Wrist	3
Knuckle	4

Table 2.6: Flap Parameters

Variable	Value	Unit
Frequency	2.11	Hz
Base AoA	7.77	deg
ω	0.11	$\sim$
Φ_1	12.24	deg
Ψ_1	15.42	deg
Ψ_2	20.00	deg
Θ_3	8.16	deg
Ψ_3	15.00	deg
Ψ_4	5.00	deg
φ_{Ψ_1}	0.38	deg
φ_{Ψ_2}	231.33	deg
φ_{Θ_3}	92.39	deg
φ_{Ψ_3}	130.44	deg
φ_{Ψ_4}	40.31	deg

Table 2.7: Critical flight properties and wing parameters

Property	Value	Units / Note
Flight speed	17	m/s
Frequency	2.66	Hz
Re	0.32	x10^6
Thickness	5	%, average on wing

The values which are substituted into Eqn 2.1 are from Sträng (2009) and are shown in Table 2.6. Further information about the flight properties are given in Table 2.7. As the airfoil is specified as being symmetric and 5% thick, the NACA 0005 was selected as the airfoil model. The SD7003 was selected as the validation airfoil due to its prominent use in numerous ornithopter analyses coupled with an abundance of available validation information. A 2D airfoil verification is viable as the employed methodologies are the same except that the 2D case does not utilize the minimal surface solver. Both the 2D and 3D cases start from position and angular data, form the chord, and add the airfoil to the chord. Then the exact same code is utilized by OpenFOAM for both cases to load the surface changes at the current simulation time. See references: Catalano and Tognaccini (2011), Ol et al. (2009), and Kang et al. (2013).

2.2 Moving Surface Code Generation

There are two primary solvers used for this project. The first is a mesh motion solver for the surface mesh of the wing. This solver is written in Python, takes the current positions of the surface mesh and transforms them to obtain the new surface geometry. The second solver, OpenFOAM, is used to solve the aerodynamics. The

integration of the two solvers is handled using a custom surface motion function within the aerodynamics solver, OpenFOAM.

The mesh motion code performs its task in several steps: surface generation, relative position determination, and surface re-mapping. The surface generation stage is the basis for the latter stages and thus shall be discussed first. Surface generation begins by calculating the joint and bone positions for the given time and flap characteristics. The non-thickened wing surface is then computed from this data. The trailing edge attachment point of the wing membrane to the body are determined by the gliding configuration and are constant for all points in the flap cycle. In other words, the chord length of the shoulder at the glide configuration determines the trailing edge attachment point of the wing and the body. The trailing edge of the wing membrane is linear between the aft body attachment point and the most distal point of the wing structure. The end is then truncated so that the minimum chord is 10 cm to mimic the configuration used by Sträng (2009). The leading edge is linear between the fore body attachment point and the point on the wing structure which has the smallest angle from the direction of travel.

2.2.1 Surface Minimization

Once the non-thickened surface is determined, it is run through a surface minimization algorithm which applies the standard minimal surface partial differential equation (PDE), Eqn 2.2, to the domain (Simon (1997)). This ensures that the surface fits tightly to all the bones and has the minimal possible surface area which is desired for wings with an elastic component. The Z term is the z location corresponding to the local non-thickened mesh. Thus, when the PDE is solved, it is expected that some of the z-coordinates of the points may change to satisfy the minimal surface condition; however, the x- and y-coordinates cannot change.

The equation is elliptical in nature and satisfies the conditions of vanishing mean curvature and minimal surface area for a given set of boundary conditions

(Simon (1997)). Due to its elliptical nature, it can readily be solved using standard techniques. As the wing shape changes with time and the input parameters can be changed, it is logical to use an approach that can accommodate these changes. Finite difference methods are well suited to perform tasks such as these as they can operate independently of their solution space; i.e. changing the shape of the domain they are solving for does not mean that different equations must be implemented, merely different values.

$$(1 + Z_x^2)Z_{yy} - 2Z_xZ_yZ_{xy} + (1 + Z_y^2)Z_{xx} = 0 \tag{2.2}$$

Finite Difference Method

Equation 2.2 is solved by applying a finite difference method with a 2nd order of magnitude. Additional orders of magnitude were considered but the 2nd order solutions yield the best results. The finite difference method is a process in which the derivative at a point is calculated based on the values at the points around it. This allows for the determination of multiple derivatives at a point given the points in its vicinity. The equations are greatly simplified when the distance between points is uniform, as is the case in a checkerboard pattern, and is typified in Figure 2.3. The uniform distance between points is set to 1 mm for this program to ensure adequate leading and trailing edge resolution. The primary equation, Eqn 2.2, can be readily decomposed into the equations shown in Eqn set 2.3. Within this equation set, there are terms of the following form: (X/Y)#(M/P), the M/P and X/Y stand for Minus/Plus and X-direction/Y-direction, respectively, while the # indicates the number of steps in that direction. These values can be stacked, one for X and one for Y within a given entry. Note that the y-direction is the spanwise direction, with 0 at the root and the x-direction is the chordwise direction with 0 at the location of the shoulder joint and positive aft. The index of the column in the matrix is modified by the (X/Y)#(M/P) value. More conventional notations are not viable due to the need to define a point as being -2 in the y- direction while also being +1 in the x-

	0	1	2	3	4	5	6	7	8	9	
0	True	True	True	True	True	True	True	True	True	True	T
1	True	True	True	True	True	True	True	True	True	True	T
2	False	True	True	True	True	True	True	True	True	True	T
3	False	False	False	False	False	False	True	True	True	True	T
4	False	False	False	False	False	False	False	False	False	False	Fa
5	False	False	False	False	False	False	False	False	False	False	Fa
6	False	False	False	False	False	False	False	False	False	False	Fa
7	False	False	False	False	False	False	False	False	False	False	Fa
8	False	False	False	False	False	False	False	False	False	False	Fa
9	False	False	False	False	False	False	False	False	False	False	Fa
10	False	False	False	False	False	False	False	False	False	False	Fa

Figure 2.3: Checkerboard pattern typical of finite difference methods. The coordinate distance is uniform between all points in both the x- and y- directions.

direction. Note that $imax$ is constant and the mesh extends beyond the borders of the wing which affects the selection criterion, as seen in Figure 2.4. Thus, an XM term is in the matrix at $k-1$ where k is the index of the active term while a YP term has the index of $k+imax$ where $imax$ is the number of terms in the chordwise direction and $YPXM$ has an index of $k+imax-1$.

$$\begin{aligned} P1 &= (1+Z_x^2)*Z_{yy} \\ P2 &= 2*Z_x*Z_y*Z_{xy} \\ P3 &= (1+Z_y^2)*Z_{xx} \\ Z &= P1-P2+P3 \end{aligned} \tag{2.3}$$

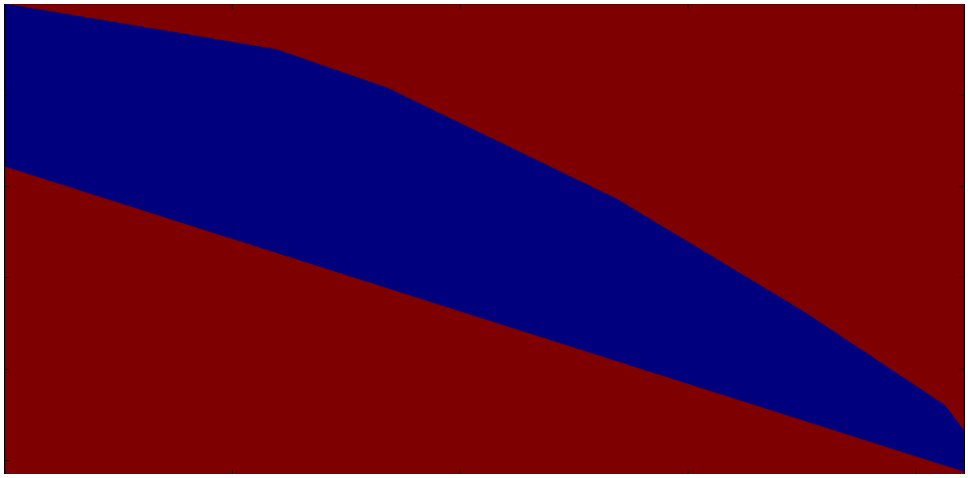


Figure 2.4: Finite difference mesh where the blue section is a part of the wing. Figure 2.3 is located at the upper left corner of this plot. The mesh is approximately 2100x1025.

The component terms are then substituted according to one of the following standard derivations, all of which are based on Taylor series expansion. There are many ways to distribute the needed points: central difference approximations utilize equivalent points on each side of the active point, forward and backward difference approximations utilize points on one side of the active point only, mixed difference approximations utilize a different non-zero number of points on each side of the active point.

Thus, for instances which require a 2nd order central difference approximation, the terms in Eqn set 2.4 are substituted. For those instances when central difference approximations are not viable, such on the boundaries, forward and backward second order approximations are used, shown in Eqn sets 2.5 and 2.6, respectively.

$$\begin{aligned} Z_x &= (XP - XM)/(2 * \Delta x) \\ Z_{xx} &= (XP - 2Z + XM)/(\Delta x^2) \\ Z_y &= (YP - YM)/(2 * \Delta y) \\ Z_{yy} &= (YP - 2Z + YM)/(\Delta y^2) \end{aligned} \tag{2.4}$$

$$\begin{aligned} Z_x &= (-3Z + 4XP - X2P)/(2 * \Delta x) \\ Z_{xx} &= (2Z - 5XP + 4X2P - X3P)/(\Delta x^2) \\ Z_y &= (-3Z + 4YP - Y2P)/(2 * \Delta y) \\ Z_{yy} &= (2Z - 5YP + 4Y2P - Y3P)/(\Delta y^2) \end{aligned} \tag{2.5}$$

$$\begin{aligned} Z_x &= (X2M - 4XM + 3Z)/(2 * \Delta x) \\ Z_{xx} &= (-X3M + 4X2M - 5XM + 2Z)/(\Delta x^2) \\ Z_y &= (Y2M - 4YM + 3Z)/(2 * \Delta y) \\ Z_{yy} &= (-Y3M + 4Y2M - 5YM + 2Z)/(\Delta y^2) \end{aligned} \tag{2.6}$$

The algorithm used to solve the minimal surface equation, Equation 2.2, is of a Newton's method configuration which utilizes a rectangular structured grid with uniform spacing as is detailed in Hoffman (2001). This method utilizes Eqn 2.7 in which $z(x, y)$ is the actual solution of the z-coordinate values of the points, $Y(x)$ is the approximate solution, and $\eta(x, y)$ is a small perturbation. By utilizing Eqn 2.8, wherein k is the iteration number, the solution reaches convergence when the $\eta(x, y)$ value is sufficiently low. The $\eta(x, y)$ value is determined by solving the matrix for the term labeled *Res* in the equations detailed in Appendix B and is equal to the local η value. Once the η value is determined, it is added to the current Z value; then, if the residual is still too high, the solution is run again.

$$z(x, y) = Z(x, y) + \eta(x, y) \tag{2.7}$$

$$Z(x, y)^{k+1} = z(x, y)^{k+1} = Z(x, y)^k + \eta(x, y)^k \tag{2.8}$$

The boundary conditions are set such that: the known bone locations are specified directly, the trailing edge is set as linear between the wing tip and the root trailing edge location, and the leading edge is set either by the bone or as linear between the root leading edge and the 4th joint, depending on location.

Newtons method solvers require initial values from which to develop the η values. This was accomplished by performing a first order linear approximation along the chordwise direction of the wing. Thus, the fore panel is linear between the leading edge and the bone; while the main body of the wing is linear from the bone to the trailing edge. As there are no discontinuities in the bone positions, the linear approximations are continuous. Furthermore, only the points on the bones have any opportunity to encounter derivative discontinuities and those points are already set by the boundary condition. From these initial estimates of z position, the Newton's method solver of the minimal surface equation can be run.

Due to the restrictions imposed by the local mesh, there are a variety different solutions to the core function which must be generated and applied. These stencils are shown in Figure 2.5. Each of these solution configurations yields its own set of equations, for details see Appendix B. Each equation is transferred to the code and checked for accuracy after having been solved. However, the presence of the equations alone does not bear fruit until the correct configuration is selected. To select the configurations, conditional statements were used to check which parts of the grid are located on the wing. That information is then used to multiply a variety of Boolean strings. The first item in the resulting string which returns true is the equation set which is used to perform the calculations. For instance, if the active point only has 1 point on the root side and 0 points on the tip side while having 0 points towards the trailing edge and several hundred points towards the leading edge: stencil G will be selected and those equations will be implemented for that line of the matrix.

The total time spent to solve the entire matrix serially is excessive; as such, the code is parallelized. During parallelization, the entire grid is broken into parts (one

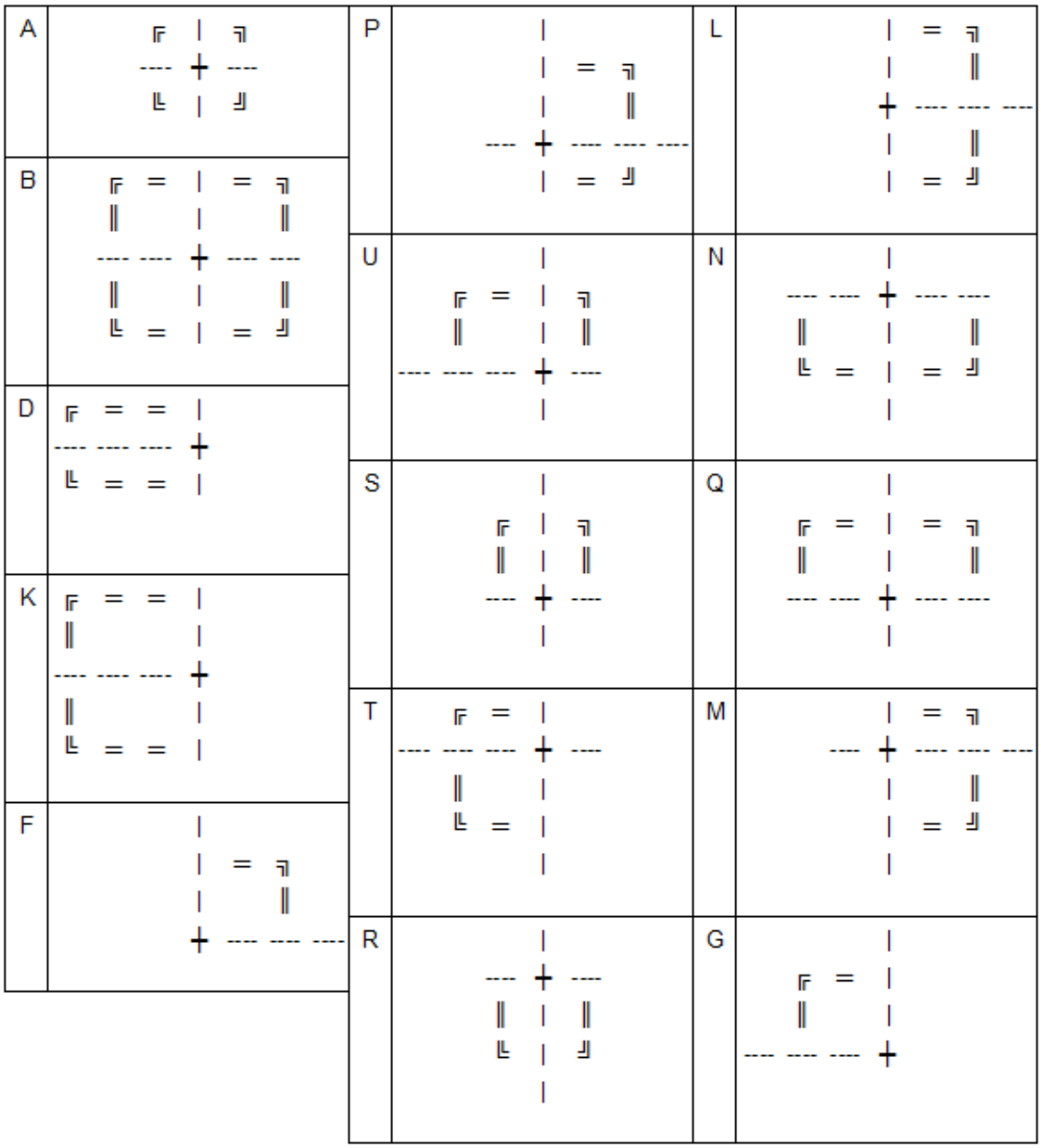


Figure 2.5: Stencil - Linearization Index Key. The cross indicates the point of interest. The single lines indicate the linear extends while the double lines indicate the range of the stencil in other directions. Note that while 4th order accuracy was considered, 2nd order is implemented.

for each available processor), each of which is given approximately the same number of points on the wing. These balanced sections are then distributed to the workers which generate their own segments of the matrix. Upon worker completion, the matrix segments are reassembled and solved. This is an iterative process until the residual of the matrix is sufficiently low. In this manner the computational time has been greatly reduced, as the time to solve the matrix is significantly less than the time spent constructing the matrix serially. Obtaining the genuine minimal surface does not significantly improve the surface given the nature of the wing model being used as the residual after the first iteration is typically on the order of 10^{-5}. Due to this low initial residual, this optimization step can often be skipped during the calculations.

2.2.2 Thickening

Thickening is the next operation to be performed on the surface. The normal vectors at each point on the non-thickened surface are determined by calculating the normal vector of each face then adding that normal to each of the vertices which comprise the face. Due to the square structured nature of the surface points, independent weighting of the vertex normals is not required. The thickness is then calculated given that the airfoil is a NACA 0005 and is applied to the normals to obtain the upper and lower surfaces of the wing.

The NACA series of airfoils are equation based shapes. The NACA 4-series symmetrical airfoils are described in equation 2.9 as given by Ladson et al. (1996), wherein d is the distance from the chord-line, c is the chord length, and t is the maximum thickness. For the NACA 0005, t is 5% of the local chord length.

$$d = 5tc(0.2969(\frac{x}{c})^{\frac{1}{2}} - 0.1260(\frac{x}{c}) - 0.3516(\frac{x}{c})^2 + 0.2843(\frac{x}{c})^3 - 0.1015(\frac{x}{c})^4) \quad (2.9)$$

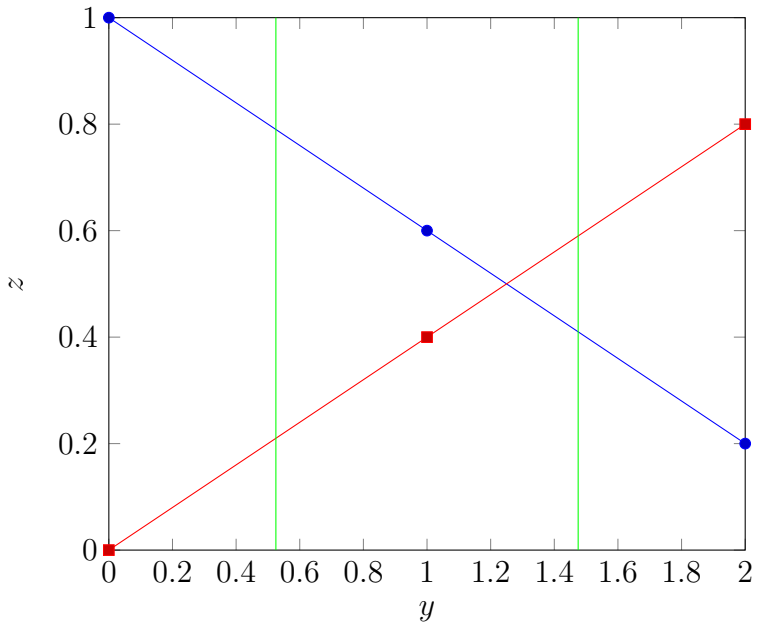


Figure 2.6: Example of column based smoothing. For the points at y=1, the column selects all points within 0.475 (0.95 / 2) of 1, denoted by the green lines. Then, if this is the upper surface, the blue point is kept while the red point is discarded. Similarly, for the points within the y=0 column, the blue point is kept for the upper surface as is the red point for the y=2 column.

Column based smoothing is then performed to remove overlapping points which are generated by angles of less than 180^o. The columns are square, centered on the active point, and have a side width of 95% of the base Δ value of 1 mm. This is demonstrated in Figure 2.6 in a 2D format. The genuine implementation is 3D yet the image is identical with only the x-axis label changing to "x". For simplicity when calculating the finite difference approximations, the Δ values in the x and y directions are identical. It is then ensured that only the outermost point remains on each surface within each column. With the column size set to slightly less than the structured grid spacing, maximum point density is ensured while removing all points which are not on the exterior of the surface.

2.2.3 Implementation

Determining the relative positions of each of the input points on the wing only has to be done once due to the fact that the output points are at identical relative positions to the output points; hence, across time steps the relative location does not change; even if the span increases, the non-dimensional nature of the relative positions handles it without incident. These relative positions are specified in terms of the chord fraction of the base point on the non-thickened surface, the spanwise fraction of the base point, and the surface upon which the point resides. The tip is slightly more complex as there can be points between the upper and lower wing surfaces. Resolving this is accomplished by allowing fractional surface selection values coupled with the known normals through the points on the non-thickened plane which reside at the tip of the wing.

New point locations are determined by performing the inverse process with a slightly different wing surface. The x and y values are readily determined as the basis for these is inherent to the overall layout of the wing. The z value determination is more complex. The points in the vicinity of the requested location are polled and weighted based on distance from the specified point of interest. These values then undergo a weighted summation process to determine the height value of the new point. This process is shown in Figure 2.8 for the wing surface generation and Figure 2.9 for the position mapping.

Upon completion of the time update cycle, the results are saved to a file for reading in by the volume mesh motion solver. Due to practical concerns (single-thread computation time), all of the result values are pre-calculated after the initial mesh is determined so that computing resources are utilized most efficiently. This process can be seen in Figure 2.7.

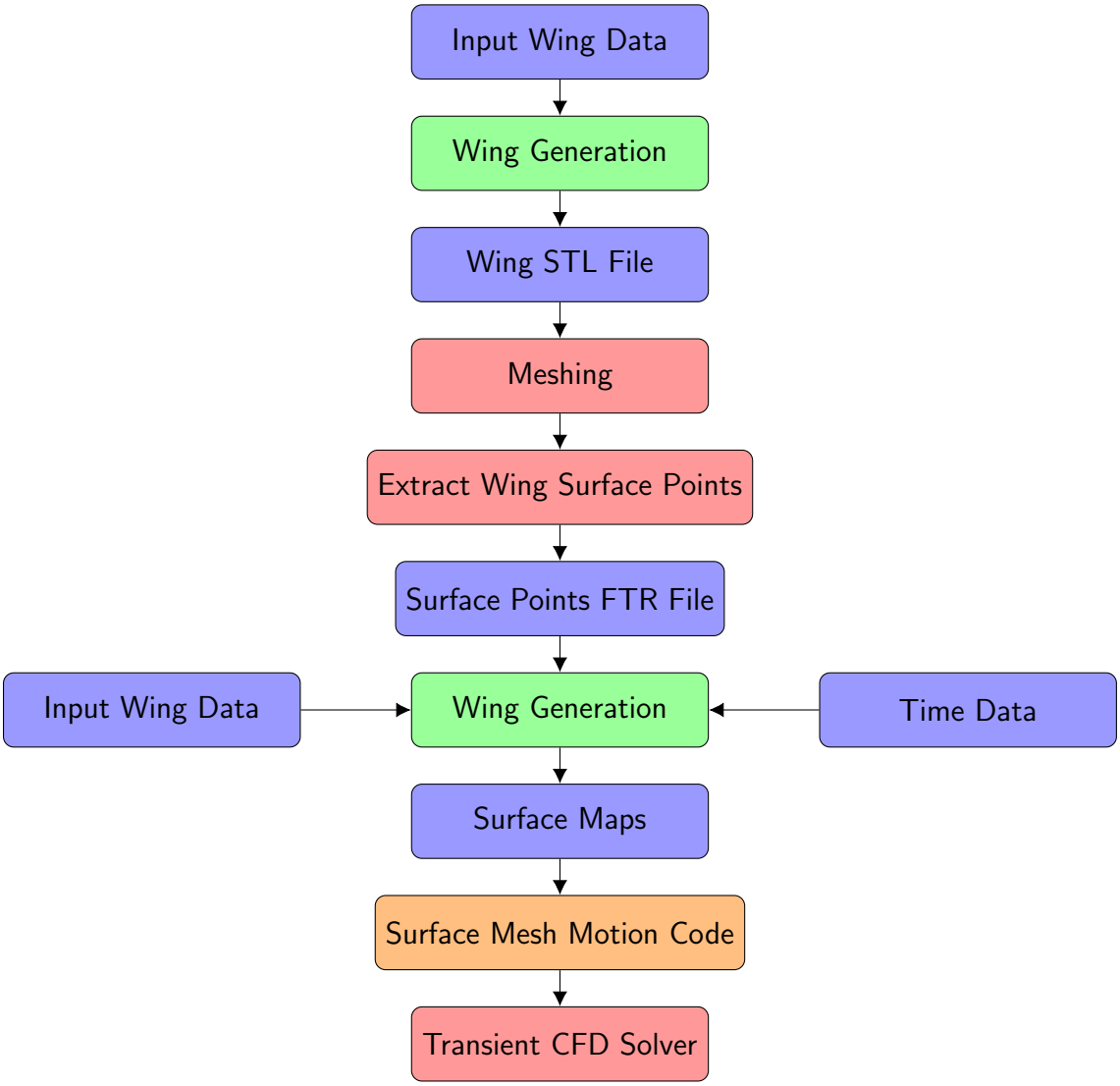


Figure 2.7: Block diagram from geometric wing data through transient CFD simulation. Green, orange, red, and blue denote python code, C^{++} code, OpenFoam code, and intermediary files, respectively.

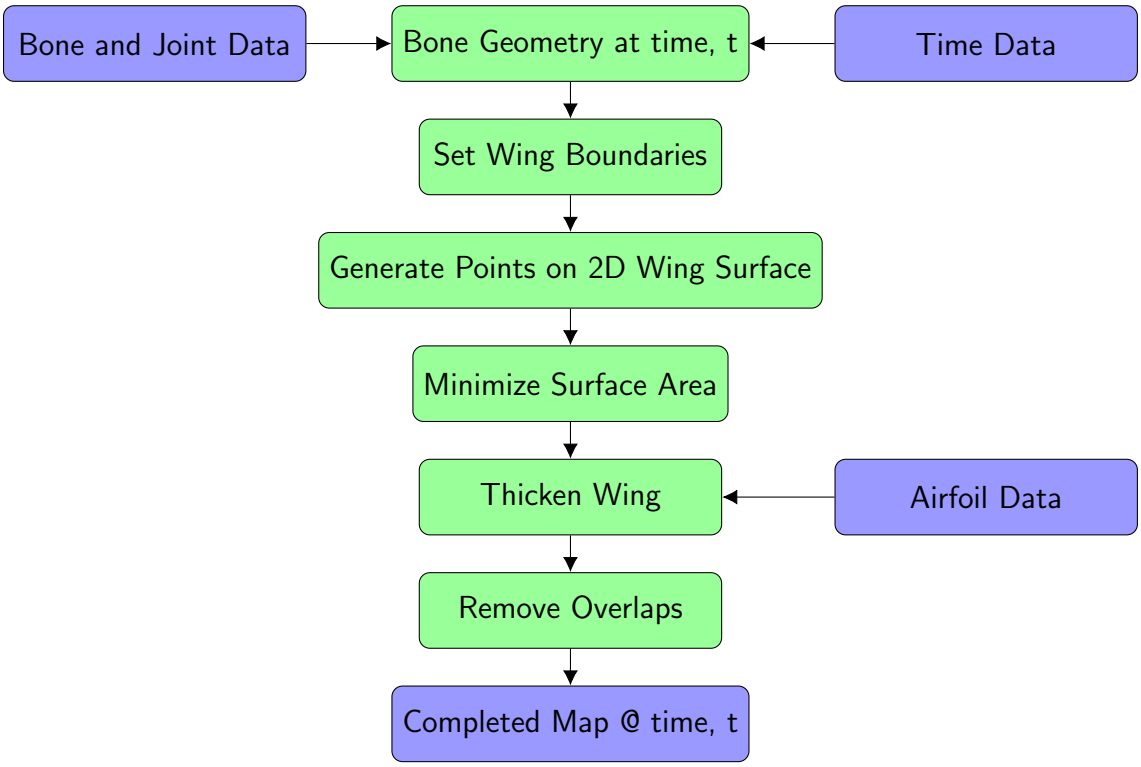


Figure 2.8: Block diagram of wing surface generation at time, t

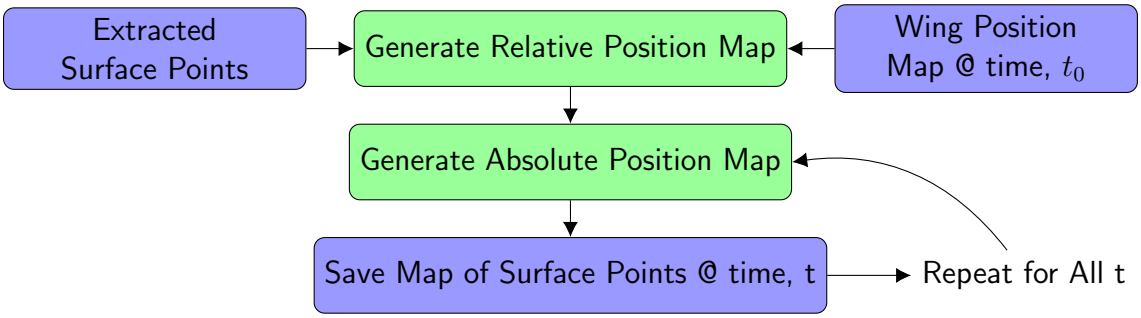


Figure 2.9: Block diagram of wing point generation at time, t

2.3 Code Optimizations

The python code has several optimizations which can significantly decrease its run time. The first optimization splits the list of required surface file times evenly among a user specified number of processors. As the standard desktop computer has at least 4 available computational threads, this can easily yield notable performance improvements by merely harnessing the processing power already present and often underutilized.

The second of these optimizations is the parallelization of the matrix generator for solving the minimal surface equation. This allows the matrix to be generated at significantly higher speeds than would otherwise be possible; furthermore, as the generation takes longer than solving the matrix, this provides a significant time savings to the overall code. It may appear that this optimization clashes with the first; however, due to slightly different run times of different surface instances, this optimization often reaps greater benefits than expected for utilizing hyper-threading (HT) cores due to the inefficient nature of adding values to a NumPy matrix.

The third significant optimization is the saving of state information for 3D cases to an independent user specified repository. Due to the nature of how the surface is interpolated, the same state point-surface can be utilized for any specific point-surface (i.e. whether there are 10 points on the mesh surface or 10 million points, they are both moved according to the same state point-surface). This optimization allows for each state point-surface to be calculated once per time per set of motion parameters rather than for every time in every simulation. As the generation of a state point-surface typically takes 30-45 minutes, this yields incredible time savings across multiple runs.

The fourth optimization is the saving of local surface deformations. As the mesh mover uses displacement rather than velocity, it is readily possible to keep the local surface deformation information independent of the time step. This allows the local surface deformation files to remain in place when the time step is changed. By keeping

these files in place, they can be skipped during the generation phase following a change in time step of the simulation. (i.e. for t=0 to t=1 for a time step of 0.5 seconds, the simulation requires 3 files: 0, 0.5, 1. When halving the time step to 0.25 seconds, the simulation required 5 files: 0, 0.25, 0.5, 0.75, 1; yet 0, 0.5, and 1 are already present and thus do not need to be recalculated.)

There are also optimizations of operation rather than only optimizations of calculation time. Among these are calls to the CFD solver files which read in the relevant information rather than having to manually keep track of and re-enter it for every run.

Naturally, there are a multitude of internal coding optimizations which assist in decreasing the run-time for every time step that is calculated. Examples of this include the choosing of correct methods to fill the surface minimization matrix, using multiplication instead of powers, etc.

2.4 CFD Solver

Foam-extend is a fork of the OpenFOAM computational fluid dynamics solver. This fork exists to give users additional tools for utilizing dynamic meshes. As the solvers are identical in function, the validation studies from one fork are often applicable to the other. Foam-extend was chosen as the fluid dynamics solver as has been validated for a variety of cases (Winter (2013), Higuera et al. (2015), and Tavangar et al. (2015)) as well as being simultaneously non-proprietary and not restricted to academia. Furthermore, this fork includes mesh motion solvers so that only the surface motions need to be input while the remainder of the mesh deformation information is automatically calculated.

2.4.1 Preparing the Wing for Mesh Generation

The python code generates the initial wing as a point cloud. As meshing programs require closed surfaces to operate correctly, the point cloud must be converted to a surface prior to meshing. This is accomplished by utilizing a program which can generate surfaces from point clouds. While performing this conversion a number of programs stated the ability to work with point clouds and surfaces; be aware that is does not mean that they have the ability to convert from a point cloud to a surface. MeshLab proved to be the most useful tool for performing this conversion (Cignoni et al. (2008)). The point cloud is easily read into the program, then several iterations of ball pivoting surface reconstruction are performed. The first iteration has the smallest ball radius and the subsequent iterations increase in radius. This method ensures that the finest details are well captured while the later iterations ensure that the surface is closed. The resulting surface is then ready to be exported in a variety of different formats.

For the 2D cases, the python code generates the surface FTR file automatically, eliminating the necessity of an external point cloud to surface conversion program.

2.4.2 Mesh Generation

The OpenFOAM suite of products includes mesh generation utilities (Ope (2016)). The most useful of these utilities are blockMesh and snappyHexMesh. As the name suggests, blockMesh allows the user to quickly and easily create a block mesh of a simple geometry. For the cases discussed here, the external geometries are all rectangular prisms. To use this utility to create the entire mesh within the rectangular domain, one must merely specify the corner points, the faces which connect them, and the number of points in each dimension. Then the meshing utility automatically performs the 1D, 2D, and 3D meshing of the volume and saves the result.

In order to remove the wing from the mesh, the snappyHexMesh utility is used. This utility refines all the cells which cross the boundary of the wing, snaps the

cells to match the contours of the surface (resulting in a body-fitted mesh), and optionally adds surface layers to the mesh. There are a multitude of options that accompany these base operations, including but not limited to: setting the cell depth for each refinement layer (i.e. how many cells "out" before a courser cell can be used), a variety of settings for snapping the local mesh to the surface of interest, growth and relative size specifications for the surface layers, and a multitude of settings for mesh quality. Furthermore, snappyHexMesh will output a variety of elements, including: hexahedrons, tetrahedrons, tetrahedral wedges, pyramids, and other polyhedrons. The discerning reader may note that these are all 3D elements, this is because OpenFOAM does not operate on 2D meshes; all 2D simulations in OpenFOAM are single cell deep 3D cases with empty boundary conditions on the faces perpendicular to the plane of interest.

2.4.3 Dynamic Mesh Decomposition

The OpenFOAM suite utilizes domain decomposition. When performing the mesh decomposition, there are a variety of methods available to the user. These methods include a "simple" method in which the user inputs the number of domains in the x, y, and z directions then the program splits the domain appropriately. Another option is hierarchical in which the user also specifies a directionality order to the decomposition process. For those who have pre-calculated mesh maps, there is also an option to apply an input file wherein each cell has been set to a domain. The final options are all relatively similar: ptscotch, parmetis, metis, and scotch. In each of these methods, the code attempts to reduce the amount of inter-domain communications. This can be particularly important when running on a high-performance cluster where the processors are on different machines. It is also of note that mesh decomposition in OpenFOAM is a genuine decomposition; the mesh is separated into distinct files which can only "see" the neighbors of the cells within their domain. Once the decomposition is completed, it is impossible to apply anything to the entire mesh without first

reconstructing the mesh. Note that this applies only to the mesh itself and not the fields which reside atop the mesh. Ope (2016)

Due to these mesh decomposition options, the integration code can be relatively simple in that it must only export the initial surface mesh, ensure the python program has run, and import the new surface for each time step.

2.4.4 Dynamic Mesh Motion

Mesh motion is handled by the individual processor cores on their respective parts of the mesh rather than the mesh motion being performed on a "total" mesh. This is due to the simple fact that the mesh decomposition is an actual mesh decomposition rather than the more typical processor zone decompositions, as discussed in the previous section. As a result, each processor only has access to its portion of the mesh and cannot access the mesh from other processor domains; however, there are tools to allow fields within the mesh to be transfered and accessed by other processors.

There are several distinct mesh motion flavors: 6 DoF rigid body motion, finite volume mesh motion, finite area surface motion, radial basis function (RBF) mesh motion, mesquite mesh motion solvers, and fluid-structure interaction (FSI) mesh motion solvers (Ope (2016)). Each type of solver has its own advantages and disadvantages.

The 6 DoF rigid body motion solvers are only viable when the entire body is solid and rigid. The standard solvers include options for periodic linear motion, periodic angular motion, and combinations thereof. There are more advanced solvers which allow for constraints of various types to be utilized and thus allow the flow to define the body motion. However, as the entire body moves as a single rigid entity, the simple motion data can be shared among the various processing domains and the mesh can be deformed without issue. The primary limitation of this solver is that the body MUST be solid and rigid.

The finite volume mesh motion (fvMotion) solvers are more free-form. These solvers allow for the mesh to be moved arbitrarily at each point. However, the mesh motion data cannot be shared between processor domains and thus this class of motion solver often encounters issues at the interfaces between processor domains in the form of points not moving as they should or face area mis-matches.

The finite area mesh motion solvers are only for surfaces but operate in a manner very similar to the finite volume mesh movers.

RBF mesh motion solvers are capable of handling very high amounts of mesh motion without issue (Bos et al. (2013)). Furthermore, the surface can be deformed in an arbitrary manner. The most significant detractors of this type of mesh motion solver are the requirement that each processor domain must be in contact with the driven patch and the significant amount of time it can take to solve the motion matrix. Less significant detractors are issues where differential mesh motion can cause the local solutions to generate face area mis-matches at the processor boundaries.

The mesquite mesh motion solver is a fairly unique type of mesh motion solver. Rather than adding a piece of code to define the patch motion, this solver is input an equation of the motion of the patch. That equation is then used to move the patch and all the other points in the domain. While this motion solver does not often encounter issues with parallel processing, it does require the patch motion to have an analytic definition.

FSI solvers are an interesting class of mesh movers in that they use the physical properties of the body to determine how the body moves and/or deforms when the fluid interacts with it. Due to the nature of this class of solvers, it is often exceedingly difficult to drive any motion of the body; instead, the user must allow the solver to handle all of the motion. These solvers rarely have issues with parallel processing; however, they do inherently increase mesh complexity and solution time due to the necessity of solving for the displacements within the body of interest.

2.4.5 Solvers

As with any CFD code, the Navier-Stokes equations are at the core of the solution method. For incompressible fluid flow, these equations can be simplified to Equation 2.10, wherein u is the velocity, t is time, P is pressure, ρ is density, and ν is the kinematic viscosity. Both the complete compressible and the reduced incompressible forms of the Navier-Stokes equations are found in a wide variety of resources, including Ope (2016). This equation is then broken down further to be used in an efficient manner by OpenFOAM. As the core functionality of the solver was not modified for this paper, its core equations will not be analyzed here. For details on the solver, please see Ope (2016).

$$\frac{\partial u}{\partial t} + u\nabla u = -\frac{\nabla P}{\rho} + \nu\nabla^2 u \tag{2.10}$$

This study utilizes a pimple solver which is designed to handle dynamic meshes and to be capable of running in parallel. It is called pimple as it is a merged PISO-Simple algorithm wherein PISO stands for pressure implicit with splitting of operator. This algorithm utilizes generic turbulence modeling, allowing for any turbulence model to be selected. A diagram of the algorithm is shown in Figure 2.10 with the PISO sub-algorithm in Figure 2.11. The sources include the OpenFOAM-extend source code (Ope (2016)), Jasak (1996), and Ferziger and Peric (2001).

As with any computational fluid dynamics study, a mesh refinement study must be performed to determine the resolution required to obtain a valid solution without wasting computational resources. Solution convergence is ensured by running the simulation through a sufficient number of cycles such that the variance from one cycle to the next is sufficiently low; furthermore, as this is a periodic temporal study, there must also be convergence within the case from one period to the next.

The variable of note in this study is the lift coefficient, C_L, which is defined in Equation 2.11, wherein ρ_∞ is the density, v_∞ is the freestream velocity, L is the lift force on the wing, and S_{ref} is the wing reference area. The force is determined by

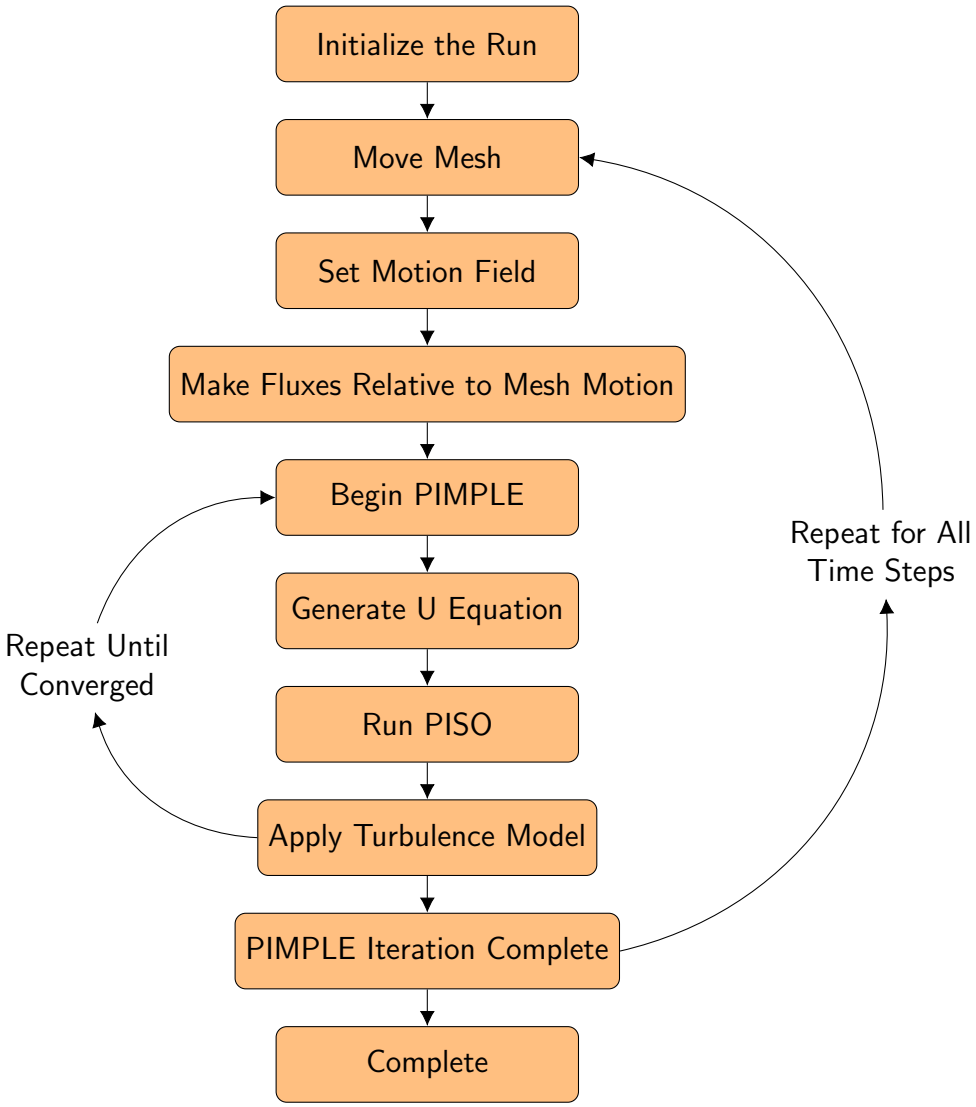


Figure 2.10: Block diagram of PimpleDyMFoam

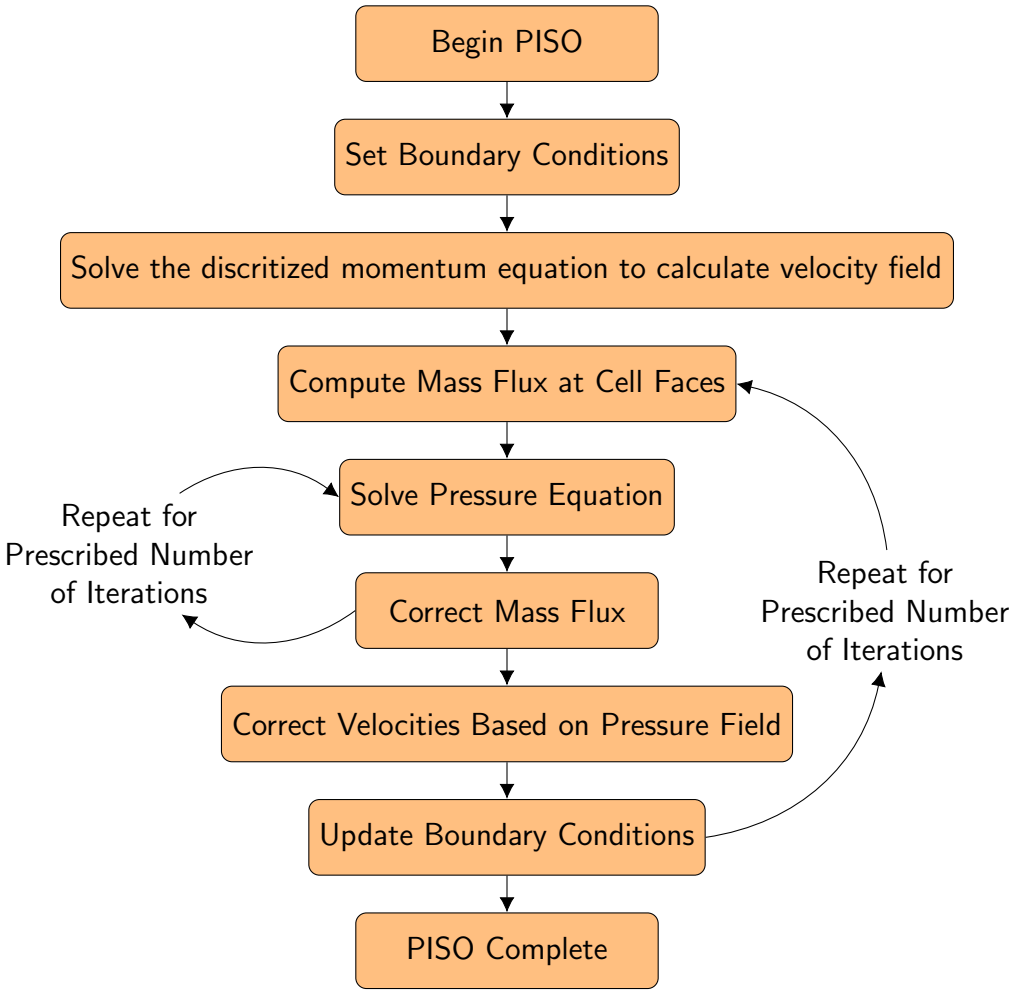


Figure 2.11: Block diagram of PISO

performing a discrete integration of the pressure field on the surface of interest in the direction of interest and is handled by the CFD solver while the other properties are defined.

$$L = \frac{1}{2}\rho_\infty v_\infty^2 C_L S_{ref} \tag{2.11}$$

Chapter 3

Code Verification

Due to the nature of new mesh motion codes, they must be validated to ensure that the results are within expected parameters. This mesh motion code is being validated for the 2D case of a pitching and plunging airfoil, an SD7003. As discussed in 2.4.2, all 2D simulations in OpenFOAM are implemented as 3D simulations which are 1 cell deep and have empty boundary conditions in the "non-existent" dimension. Employing code verification of this type is viable as both the verification code and the fully 3D code utilize the same methodologies: begin with a singular point and angle in the chordwise direction, add the chord, thicken the wing, export/import. The only methodological difference is that the pseudo-2D process does not utilize the surface minimization algorithm. The reference papers for validation are by Baik and Bernal (2012), Ol et al. (2009), and Catalano and Tognaccini (2011). In these papers, the airfoil motion is defined by equation set 3.1 with constants given in Table 3.1.

$$\begin{aligned} h &= h_0 * c * cos(2\pi * f * t) \\ \theta &= \theta_0 * cos(2\pi * f * t + \varphi) \\ f &= \frac{k * U_\infty}{\pi * c} \end{aligned} \tag{3.1}$$

By applying these equations, point locations for the surface at each time step are generated and the validation case is run.

Table 3.1: Validation motion constants

Parameter	Value
k	0.25
φ	$\frac{\pi}{2}$
h_0	0.5
θ_0	8.42 deg

The validation case is run as a Reynolds Averaged Simulation (RAS) utilizing a $k-\omega$ SST turbulence model. This model utilizes the equations shown in section A.2.

A different validation case was run as a RAS utilizing the Spalart-Allmaras (SA) turbulence model. Details on this turbulence model can be found in section A.3. Upon running, it has been found that the SA model sheds much more readily than the $k-\omega$ SST model and has thus been removed due to excessive resource usage caused by the higher required refinement levels when compared to the $k-\omega$ SST case.

3.1 Mesh Convergence Study

The mesh convergence study is performed by using a set far-field mesh and modifying the refinement level required at the wing surface. The study is continued until the C_L value changes by less than 2%. Once convergence of the static mesh is attained, the same mesh is used for the dynamic analysis. As can be seen in Figure 3.1, there is less than a 2% change in C_L value between the 4th and 6th refinement levels, corresponding to the highest numbers of cells in the plot. Thus, the 4th refinement level, corresponding to the 2nd point from the left, is considered converged. See Section 2.4.2 for details on the mesh generation process.

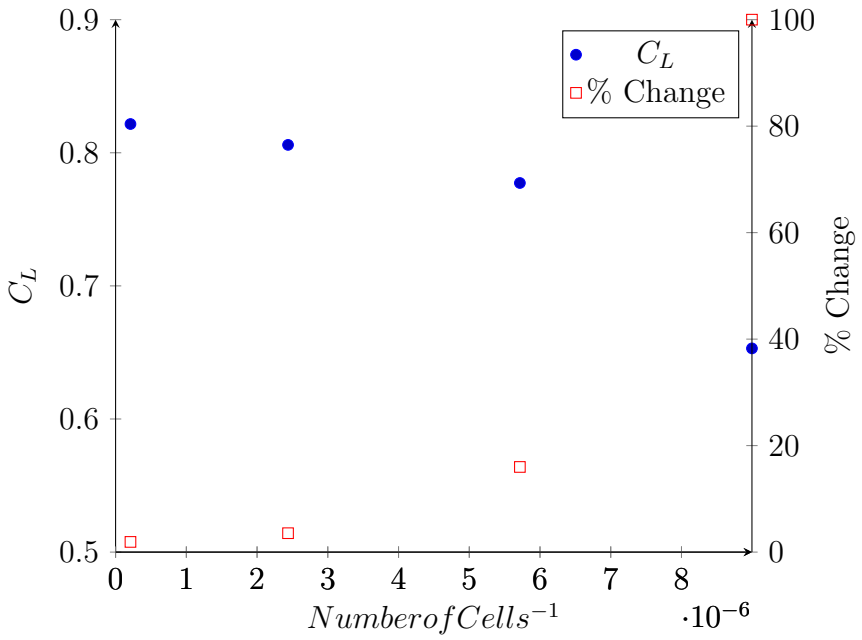


Figure 3.1: 2D C_L validation comparison

Additionally, the pressure distribution is reasonably accurate. In the paper by Catalano and Tognaccini (2011), the authors discuss how the $k-\omega$ SST model has difficulties locating the transition point of this particular flow. With this information it is remarkable how well the rest of the pressure values agree, see Figure 3.2.

Thus, the resulting mesh has a far field point every 0.1 meters on a box which is 21x21 meters; resulting in a far field box of 210 x 210. On the wing surface, that 0.1 meter distance has been reduced by half a total of 4 times; resulting in an average distance of 1/16 meter between the points and a total of 628 points on the wing surface.

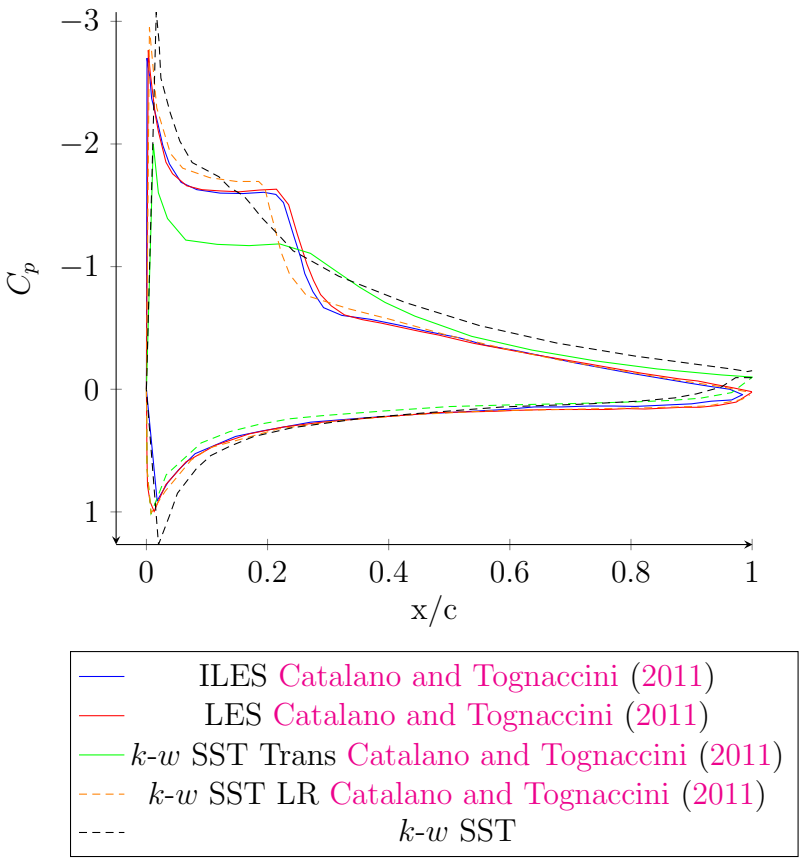


Figure 3.2: 2D C_p comparison. All k-w runs are steady-state solutions. The LES runs are time averaged.

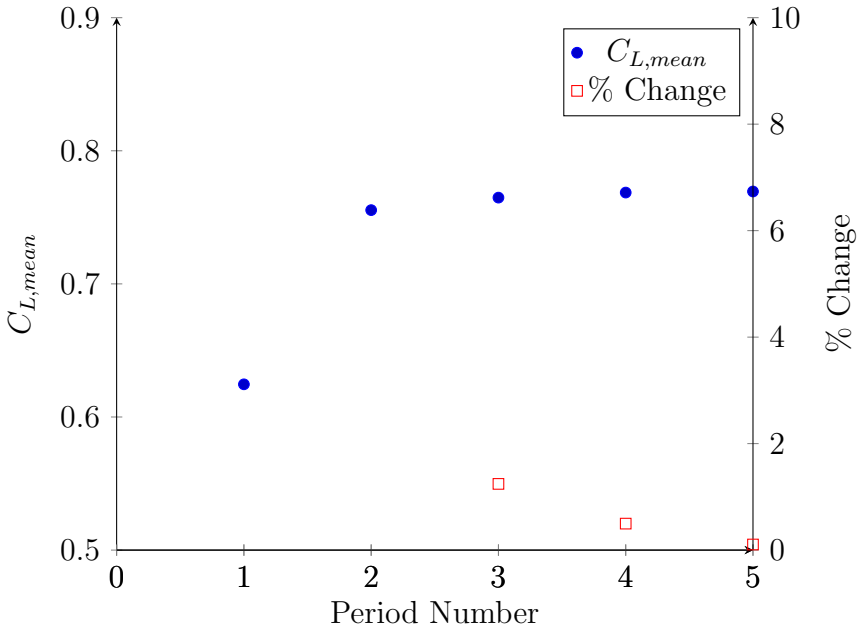


Figure 3.3: 2D $C_{L,mean}$ periodic convergence

3.1.1 Temporal Convergence

The dynamic simulation is run using the same mesh as used for the static mesh convergence analysis. The results from the static analysis are the initial conditions for the dynamic case to enhance the convergence rate. The time step is set such that convergence rate is optimized, in this case such that the initial maximum mesh Courant number is $\sim$0.15.

The temporal convergence is determined by checking when the average and maximum C_L values change by less than 1% compared to the previous period. As shown in Figure 3.3, it can be seen that the solution is converged within 4 iterations.

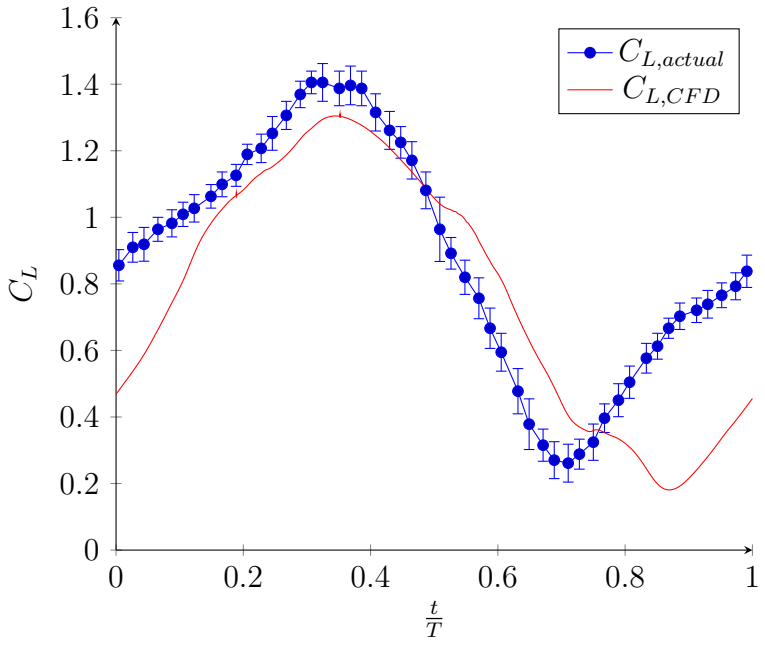
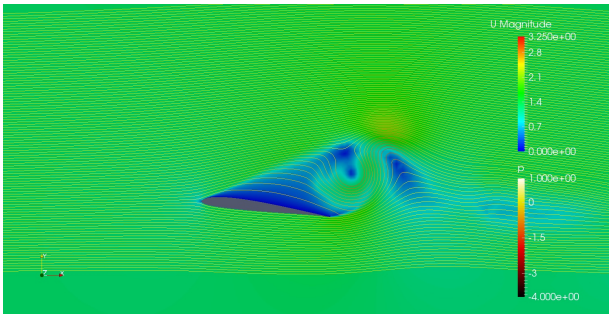


Figure 3.4: C_L Validation Comparison. $C_{L,actual}$ is from Ol et al. (2009).

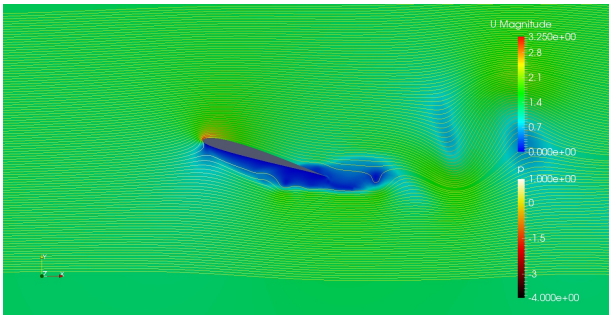
3.2 Results

The solution is presented in Figure 3.4 with the results from Ol et al. (2009). Additional data from Ol et al. (2009) is presented and compared with the CFD results in Table 3.2. Figure 3.5 shows the CFD results at selected times. Higher resolution plots of the images presented in Figure 3.5 are shown in Section C.1. The only odd behavior is the delayed reaction of the minimum C_L; this is likely caused by the boundary layer transition and separation issues mentioned in the paper by Catalano and Tognaccini (2011). The lower edge vortex shedding is indicated in Figure 3.6 and supports this claim. These vortexes can also be seen in part b of Figure 3.5. No frequency analysis was performed to ensure that no other factors are significant as the purpose of the verification is to show that the motion code functions correctly.

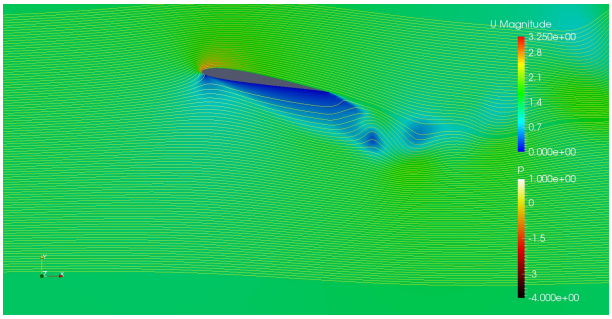
a)



b)



c)



d)

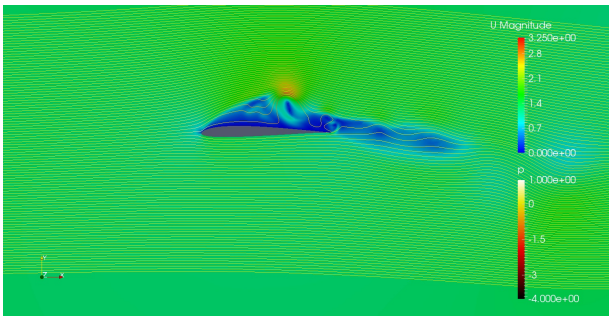


Figure 3.5: 2D validation images with t/T = 0.00, 0.25, 0.50, 0.75 for a, b, c, and d, respectively, with the streamlines colored based on pressure and the background color based on the velocity magnitude

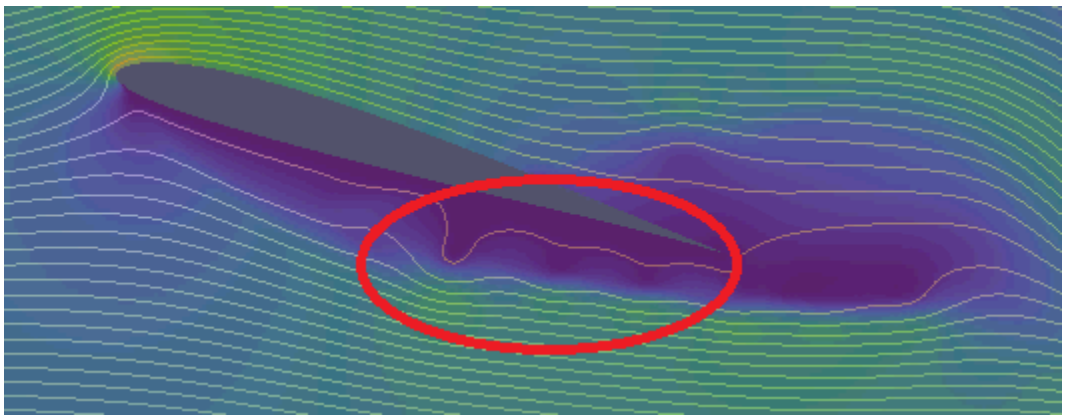


Figure 3.6: 2D CFD results with lower edge vortex shedding circled in red

Table 3.2: Additional Validation Data. "Actual" values are from Kang et al. (2013)

Source	$C_{L,mean}$	$C_{L,max}$
Actual	0.89	1.34
Calculated	0.77	1.33

Chapter 4

3D Results

4.1 Surface Motion

The motion of the 3D multi-jointed wing for *Coloborhynchus robustus* as calculated in the J5 case by Sträng (2009) is shown in Figures 4.1 and 4.2. As is typical for ornithopters, the wing tip follows a roughly elliptical path through space, relative to the body. Higher resolution orthogonal images are available in Section C.2.

The wing tip was clipped to obtain a minimum chord of 10cm, as specified by Sträng (2009). This clipping is performed while the wing is in the glide configuration, which is not present during the flap pattern. As a result, the clipped edge is rarely aligned with the flow during the motion cycle.

During the latter part of the flap cycle, the elbow joint causes the apparent wing area from the Z+ view to shrink considerably; however, there is also a significant amount of twisting occurring which is the actual reason for the increase in apparent aspect ratio. Confirmation of this is provided by analyzing the motion from the X+ and Y+ views.

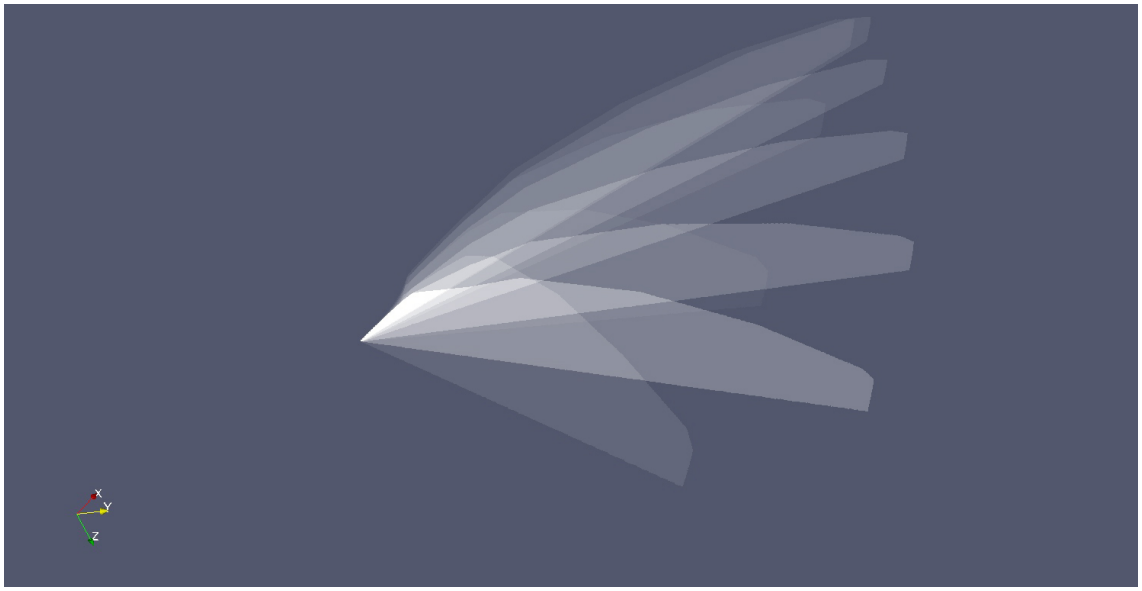


Figure 4.1: Isometric of 3D wing motion looking towards body

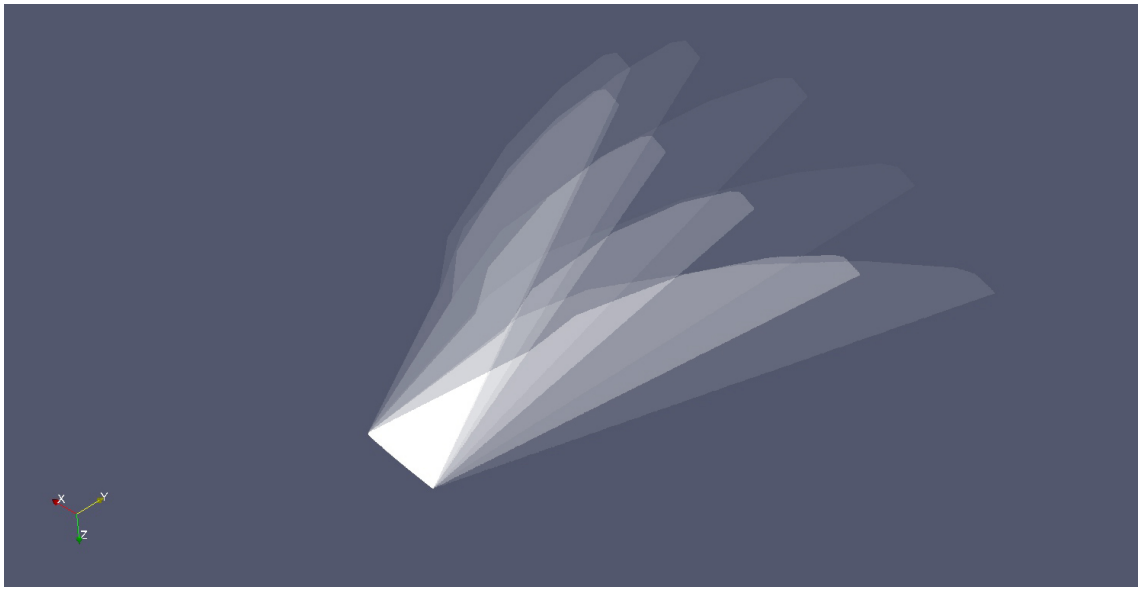


Figure 4.2: Isometric of 3D wing motion from above body

4.2 3D CFD Solution Attempts

The custom code accurately models the motion of the Pterosaur wing, as seen in Figure 4.1; however, there were several unforeseen issues with implementation in the CFD solver. The primary issue is the inability of the mesh motion solvers to handle the surface motion required by the wing surface model. The fvMotion solvers were unable to deal with any 3D motion that was not completely uniform. As a result of this issue, the RBF motion functions were implemented instead.

Yet even the RBF motion solvers proved to be unable to handle the motion required in a reasonable amount of time. Due to the nature of the RBF solvers coupled with the surface motion behavior, no coarsening of the surface could be implemented resulting in a very large motion matrix. The large matrix issue is exacerbated by the restrictions imposed by the meshing utility, snappyHexMesh. In this utility, for the cells to properly snap to the surface, the cell size must be sufficiently small. Yet, as the mesher performs isotropic refinement, the mesh contains an extremely high number of cells on the wing surface. As each vertex is a control point for the RBF motion solver, the matrix becomes so large that it cannot be solved within an hour on a 3.2 GHz machine which is not RAM limited.

The next logical course of action involves attempting to run the simulation in another program; in this instance, SU2 was selected as the attempted simulation program. The first step in this process is converting the mesh to a form usable by SU2. Yet converting the mesh proved to be more difficult than anticipated. The OpenFOAM suite of solvers can handle much more diverse cell types (hexahedrons, tetrahedrons, tetrahedral wedges, etc) than SU2, as discussed in 2.4.2; as such, direct mesh conversion was not possible. Thus, the original STL file was imported in PointWise to create a new mesh. Unfortunately, PointWise cannot properly import the MeshLab generated STL despite the fact that numerous other program can import it without issue. As a result, no new 3D mesh could be generated in a timely manner for use in SU2.

Due to these difficulties, the 3D CFD solution for this prescribed surface motion is unobtainable at this time. Thus, this paper discusses the design and implementation of the surface mover and an overview of the ways it was implemented so that the code can be utilized in the future. Furthermore, the 2D validation shows that the method and implementation are fully viable for the 2D case at the present time while Figure 4.1 demonstrates the validity of the 3D wing surface mover.

Chapter 5

Conclusions

The 2D results demonstrate that the wing motion code is viable for arbitrary wing motion. However, the mesh movers for the 3D case are not yet capable of coping with the high cell count inherent to the isotropic meshes generated in OpenFOAM. As discussed in Chapter 4, there were several issues which precluded the possibility of obtaining results for the high fidelity 3D wing model. However, the function which imports the moving surface into the OpenFOAM mesh has been validated by the 2D case, as OpenFOAM treats all 2D cases as 3D cases with special boundary conditions. Furthermore, the methodology of the 2D and 3D surface movers are quite similar in how they process the input points and return the new set of points for the desired time. The surface mover for the 3D case generates the appropriate surfaces correctly for any given time, as seen in Figure 4.1; thus, the primary issue which prevents a 3D result is the lack of a mesh mover which can handle the surface motion in a timely and parallel manner. Once a suitable mesh mover is found, the code presented in this paper can be applied to generate the high-fidelity solution requested for any multi-jointed wing with multiple degrees of freedom.

5.1 Further Work

Once a suitable CFD solver with a viable parallel mesh motion algorithm is identified, the 3D case can be run in full detail. This will allow for a basic comparison of the force disparities between the high and low fidelity models. Upon comparing the difference ratio at these Reynold's numbers with those obtained from other studied for significantly smaller craft, it should be possible to determine an empirical relationship for the disparity ratio vs Reynold's number. This information would be quite useful as a correction factor for low fidelity analysis as it would allow for results significantly closer to the actual values while maintaining the speed and convenience of low fidelity modeling.

Bibliography

(2016). *OpenFOAM User Guide version 4.0*. The OpenFOAM Foundation. 25, 27, 29

Baik, Y. S. and Bernal, L. P. (2012). Experimental study of pitching and plunging airfoils. *Exp Fluids*, pages 1922–1979. 33

Bos, F. M., van Oudheusden, B. W., and Bijl, H. (2013). Radial basis function based mesh deformation applied to simulation of flow around flapping wings. *Computers Fluids*, 79:167 – 177. 28

Catalano, P. and Tognaccini, R. (2011). Rans analysis of the low-reynolds number flow around the sd7003 airfoil. *Aerospace Science and Technology*, 15(8):615 – 626. 10, 33, 35, 36, 38

Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., and Ranzuglia, G. (2008). MeshLab: an Open-Source Mesh Processing Tool. In Scarano, V., Chiara, R. D., and Erra, U., editors, *Eurographics Italian Chapter Conference*. The Eurographics Association. 25

Dacles-Mariani, J., Zilliac, G. G., Chow, J. S., and Bradshaw, P. (1995). Numerical/experimental study of a wingtip vortex in the near field. *AIAA Journal*, 33:1561–1568. 56

de Ricqlès, A. J., Padian, K., Horner, J. R., and Francillon-Viellot, H. (2000). Palaeohistology of the bones of pterosaurs (reptilia: Archosauria): anatomy, ontogeny, and biomechanical implications. *Zoological Journal of the Linnean Society*, 129:349–385. 2

Ferziger, J. H. and Peric, M. (2001). *Computational Methods for Fluid Dynamics*. Springer. 3rd Ed. 29

Gopalakrishnan, P. (2008). *Unsteady Aerodynamic and Aeroelastic Analysis of Flapping Flight*. Virginia Polytechnic Institute. Ph.D. Thesis. 1

Hall, K. C., Pigott, S. A., and Hall, S. R. (1998). Power requirements for large-amplitude flapping flight. *Journal of Aircraft*, 35(3):352–361. 1

Higuera, P., Losada, I. J., and Lara, J. L. (2015). Three-dimensional numerical wave generation with moving boundaries. *Coastal Engineering*, 101:35–47. 24

Hoffman, J. D. (2001). *Numerical Methods for Engineers and Scientists, Second Edition*. CRC Press. 15

Jasak, H. (1996). *Error Analysis and Estimation for the Finite Volume Method with Applications to Fluid Flows*. Imperial College, London. Ph.D. Thesis. 29

Jones, M. A. (2013). *CFD Analysis and Design Optimization of Flapping Wing Flows*. North Carolina A&T State University. Ph.D. Thesis. 2

Kang, C.-k., Aono, H., Baik, Y. S., Bernal, L. P., and Shyy, W. (2013). Fluid dynamics of pitching and plunging flat plate at intermediate reynolds numbers. *AIAA Journal*, 21(2):315–329. x, 10, 40

Kellner, A. W. and Tomida, Y. (2000). Description of a new species of anhangueridae (pterodactyloidea) with comments on the pterosaur fauna from the santana formation (aptian-albian), northeastern brazil. *National Science Museum Monographs*, 17:ix–137. xi, 5

Ladson, C. L., Jr., C. W. B., Hill, A. S., and Sproles, D. W. (1996). Computer program to obtain ordinated for naca airfoils. *NASA Technical Memorandum 4741*. 18

Menter, F. R. (1993). Zonal two equation k- turbulence models for aerodynamic flows. *24th Fluid Dynamics Conference*. AIAA Paper 93-2906. 54

Menter, F. R. (1994). Two-equation eddy-viscosity turbulence models for engineering applications. *AIAA Journal*, 32(8):1598–1605. 54

Ol, M. V., Bernal, L., Kang, C.-K., and Shyy, W. (2009). Shallow and deep dynamic stall for flapping low reynolds number airfoils. *Experiments in Fluids*, 46(5):883–901. xii, 10, 33, 38

Pearson, P.-O., Willis, D., and Peraire, J. (2011). Numerical simulation of flapping wings using a panel method and a high-order navier-stokes solver. *International Journal for Numerical Methods in Engineering*, 01:1–20. 2

Reynolds, O. (1995). On the dynamical theory of incompressible viscous fluids and the determination of the criterion. *Proceedings: Mathematical and Physical Sciences*, 451(1941):5–47. 54

Simon, L. (1997). *Geometry V*. Springer Berlin Heidelberg. 11, 12

Spalart, P. R. and Allmaras, S. R. (1992). A one-equation turbulence model for aerodynamic flows. *30th Aerospace Sciences Meeting and Exhibit*. AIAA Paper 92-0439. 56

Spalart, P. R. and Allmaras, S. R. (1994). A one-equation turbulence model for aerodynamic flows. *La Recherche Aerospatiale*, (1):5–21. 56

Spedding, G., Hedenström, A., and Rosén, M. (2003). Quantitative studies of the wakes of freely flying birds in a low-turbulence wind tunnel. *Experiments in Fluids*, 34:291–303. 1

Spedding, G. R. (1987). The wake of a kestrel (falco tinnunculus) in gliding flight. *Journal of Experimental Biology*, 127:45–57. 1

Sträng, A. K. (2009). *Efficient Flapping Flight of Pterosaurs*. Stanford University. Ph.D. Thesis. x, xi, xvii, 2, 3, 4, 5, 6, 7, 8, 10, 11, 41

Tavangar, S., Hashemabadi, S. H., and Saberimoghadam, A. (2015). Cfd simulation for secondary breakup of coalwater slurry drops using openfoam. *Fuel Processing Technology*, 132:153 – 163. 24

Wilkinson, M. T. (2008). Three-dimensional geometry of a pterosaur wing skeleton, and its implications for aerial and terrestrial locomotion. *Zoological Journal of the Linnean Society*, 154:27–69. 2

Winter, M. (2013). *Benchmark and validation of Open Source CFD codes, with focus on compressible and rotating capabilities, for integration on the SimScale platform.* Chalmers University of Technology. Master's Thesis. 24

Witton, M. P. and Habib, M. B. (2010). On the size and flight diversity of giant pterosaurs, the use of birds as pterosaur analogues and comments on pterosaur flightlessness. *PLOS ONE*, 5(11):1–18. 2

Appendix

Appendix A

Summary of Equations

A.1 RAS

Reynold's Averaged Simulation (RAS) or Reynold's Averaged Navier-Stokes (RANS), was first developed in 1894 by Reynolds (1995). During their development, these equations replace the time dependence of the incompressible Navier-Stokes equations with "variance" terms. Thus each actual velocity (u) is comprised of a mean velocity ($\overline{u}$) and a time-variable velocity (u').

The derivation can be followed in the original paper by Reynolds (1995) or in numerous other sources. This technique is only viable for incompressible, Newtonian fluids. The final equation, which is utilized in CFD codes, is shown in A.1 and has been written in Einstein notation and for the 2D case.

$$\rho\overline{u}_j\frac{\partial\overline{u}_i}{\partial x_j} = \rho\overline{f}_i + \frac{\partial}{\partial x_j}[-\overline{p}\delta_{ij} + \mu(\frac{\partial\overline{u}_i}{\partial x_j} + \frac{\partial\overline{u}_j}{\partial x_i}) - \rho\overline{u_i'u_j'}] \tag{A.1}$$

A.2 $k-w$ SST

This model is a combination of the $k-w$ turbulence model and shear-stress transport. Hence, this is a tonal two equation model. The relevant equations and constants are shown below in A.2 through A.11 and are from Menter (1994) and Menter (1993).

Kinematic Eddy Viscosity:

$$\nu_T = \frac{a_1 k}{\max(a_1\omega, SF_2)} \tag{A.2}$$

Turbulent Kinetic Energy:

$$\frac{\partial k}{\partial t} + U_j\frac{\partial k}{\partial x_j} = P_k - \beta^* k\omega + \frac{\partial}{\partial x_j}[(\nu + \sigma_k\nu_T)\frac{\partial k}{\partial x_j}] \tag{A.3}$$

Specific Dissipation Rate:

$$\frac{\partial \omega}{\partial t} + U_j\frac{\partial \omega}{\partial x_j} = \alpha S^2 - \beta\omega^2 + \frac{\partial}{\partial x_j}[(\nu + \sigma_\omega\nu_T)\frac{\partial \omega}{\partial x_j}] + 2(1-F_1)\sigma_{\omega 2}\frac{1}{\omega}\frac{\partial k}{\partial x_i}\frac{\partial \omega}{\partial x_i} \tag{A.4}$$

Auxiliary Relations:

$$F_1 = \tanh\{\min[\max(\frac{\sqrt{k}}{\beta^*\omega y}, \frac{500\nu}{y^2\omega}), \frac{4\sigma_{\omega 2}k}{CD_{k\omega}y^2}]^4\} \tag{A.5}$$

$$F_2 = \tanh[\max(\frac{\sqrt{k}}{\beta^*\omega y}, \frac{500\nu}{y^2\omega})^2] \tag{A.6}$$

$$P_k = \min(\tau_{ij}\frac{\partial U_i}{\partial x_j}, 10\beta^* k\omega) \tag{A.7}$$

$$CD_{k\omega} = \max(2\rho\sigma_{\omega 2}\frac{1}{\omega}\frac{\partial k}{\partial x_i}\frac{\partial \omega}{\partial x_i}, 10^{-10}) \tag{A.8}$$

$$\Phi = \Phi_1 F_1 + \Phi_2(1-F_1) \tag{A.9}$$

Constants:

$$\alpha_1 = \frac{5}{9}, \quad \alpha_2 = 0.44, \quad \sigma_{k1} = 0.85, \quad \sigma_{k2} = 1 \tag{A.10}$$

$$\sigma_{\omega 1} = 0.5, \quad \sigma_{\omega 2} = 0.856, \quad \beta_1 = \frac{3}{40}, \quad \beta_2 = 0.0828, \quad \beta^* = \frac{9}{100} \tag{A.11}$$

A.3 SA

The Spalart-Allmaras (SA) model solves a single equation for a viscosity-like variable, nut. The primary equation is shown in A.12. The supporting equations are shown in A.13 through A.16 where d is the distance to the closest surface and are provided by Spalart and Allmaras (1992), Spalart and Allmaras (1994), and Dacles-Mariani et al. (1995).

$$\begin{aligned}\frac{\partial \overline{v}}{\partial t} + u_j \frac{\partial \overline{v}}{\partial x_j} = C_{b1}\left[1 - f_{t2}\right]\overline{S}\overline{v} + \frac{1}{\sigma}\{\nabla \cdot [(v+\overline{v})\nabla\overline{v}] + C_{b2}\left|\nabla\overline{v}\right|^2\} \\ - \left[C_{w1} f_w - \frac{C_{b1}}{\kappa^2} f_{t2}\right](\frac{\overline{v}}{d})^2 + f_{t1}\Delta U^2\end{aligned} \tag{A.12}$$

Where:

$$v_t = \overline{v} f_{v1}, \quad f_{v1} = \frac{X^3}{X^3 + C_{v1}^3}, \quad X := \frac{\overline{v}}{v}, \quad \overline{S} \equiv S + \frac{\overline{v}}{\kappa^2 d^2} f_{v2}, \quad f_{v2} = 1 - \frac{X}{1 + X f_{v1}} \tag{A.13}$$

and

$$\begin{aligned} S &\equiv \sqrt{2\Omega_{ij}\Omega_{ij}}, & \Omega_{ij} &\equiv \frac{1}{2}(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}) \\ f_w &= g\left[\frac{1 + C_{w3}^6}{g^6 + C_{w3}^6}\right]^{1/6}, & g &= r + C_{w2}(r^6 - r) \\ r &\equiv \frac{\overline{v}}{\overline{S}\kappa^2 d^2}, & f_{t1} &= C_{t1} g_t \exp(-C_{t2}\frac{\omega_t^2}{\Delta U^2}\left[d^2 + g_t^2 d_t^2\right]) \\ f_{t2} &= C_{t3}\exp(-C_{t4}X^2) \end{aligned} \tag{A.14}$$

Constants:

$$\sigma = 2/3, \quad C_{b1} = 0.1355, \quad C_{b2} = 0.622, \quad \kappa = 0.41, \quad C_{w1} = \frac{C_{b1}}{\kappa^2} + (\frac{1 + C_{b2}}{\sigma}) \tag{A.15}$$

$$C_{w2} = 0.3, \quad C_{w3} = 2, \quad C_{v1} = 7.1, \quad C_{t1} = 1, \quad C_{t2} = 2, \quad C_{t3} = 1.1, \quad Ct4 = 2 \tag{A.16}$$

An optional modification proposed by Spalart changes the last two values, see A.17.

$$C_{t3} = 1.2, \quad C_{t4} = 0.5 \tag{A.17}$$

The boundary condition at a wall is imposed by $\overline{v} = 0$.

Appendix B

Minimal Surface Equations

The minimal surface equations are detailed in this appendix. The first subsection details the components which are combined to form the full equations. Latter subsections display the matrix equations in detail. The key which shows the relation between the linearization index and the stencil type is shown in Figure 2.5. The linearization section has some terms which need clarification: (X/Y)#(M/P), the M/P and X/Y stand for Minus/Plus and X-direction/Y-direction, respectively, while the # indicates the number of steps in that direction. These values can be stacked, one for X and one for Y within a given entry. Note that the y-direction is the spanwise direction, with 0 at the root and the x-direction is the chordwise direction with 0 at the location of the shoulder joint and positive aft. More conventional notations are not viable due to the need to define a point as being -2 in the y- direction while also being +1 in the x- direction.

B.1 Components

2nd Order (+/- 1)

$$\begin{aligned} Z_x &= (XP - XM)/(2 * \Delta x) \\ Z_{xx} &= (XP - 2Z + XM)/(\Delta x^2) \\ Z_y &= (YP - YM)/(2 * \Delta y) \\ Z_{yy} &= (YP - 2Z + YM)/(\Delta y^2) \end{aligned} \tag{B.1}$$

4th Order (+/- 2)

$$\begin{aligned} Z_x &= (X2M - 8XM + 8XP - X2P)/(12 * \Delta x) \\ Z_{xx} &= (-X2M + 16XM - 30Z + 16XP - X2P)/(12 * \Delta x^2) \\ Z_y &= (Y2M - 8YM + 8YP - Y2P)/(12 * \Delta y) \\ Z_{yy} &= (-Y2M + 16YM - 30Z + 16YP - Y2P)/(12 * \Delta y^2) \end{aligned} \tag{B.2}$$

2nd Order (-3)

$$\begin{aligned} Z_x &= (X2M - 4XM + 3Z)/(2 * \Delta x) \\ Z_{xx} &= (-X3M + 4X2M - 5XM + 2Z)/(\Delta x^2) \\ Z_y &= (Y2M - 4YM + 3Z)/(2 * \Delta y) \\ Z_{yy} &= (-Y3M + 4Y2M - 5YM + 2Z)/(\Delta y^2) \end{aligned} \tag{B.3}$$

2nd Order (+3)

$$\begin{aligned} Z_x &= (-3Z + 4XP - X2P)/(2 * \Delta x) \\ Z_{xx} &= (2Z - 5XP + 4X2P - X3P)/(\Delta x^2) \\ Z_y &= (-3Z + 4YP - Y2P)/(2 * \Delta y) \\ Z_{yy} &= (2Z - 5YP + 4Y2P - Y3P)/(\Delta y^2) \end{aligned} \tag{B.4}$$

4th Order (-1 / +3)

$$\begin{aligned} Z_x &= (-3XM - Z + 5XP - X2P)/(6 * \Delta x) \\ Z_{xx} &= (11XM - 20Z + 6XP + 4X2P - X3P)/(11 * \Delta x^2) \\ Z_y &= (-3YM - Z + 5YP - Y2P)/(6 * \Delta y) \\ Z_{yy} &= (11YM - 20Z + 6YP + 4Y2P - Y3P)/(11 * \Delta y^2) \end{aligned} \tag{B.5}$$

4th Order (-3 / +1)

$$\begin{aligned} Z_x &= (X2M - 5XM + Z + 3XP)/(6 * \Delta x) \\ Z_{xx} &= (-X3M + 4X2M + 6XM - 20Z + 11XP)/(11 * \Delta x^2) \\ Z_y &= (Y2M - 5YM + Z + 3YP)/(6 * \Delta y) \\ Z_{yy} &= (-Y3M + 4Y2M + 6YM - 20Z + 11YP)/(11 * \Delta y^2) \end{aligned} \tag{B.6}$$

B.2 Linearization A

$$\begin{aligned} P1 &= (1 + Z_x^2) * Z_{yy} \\ P2 &= 2 * Z_x * Z_y * Z_{xy} \\ P3 &= (1 + Z_y^2) * Z_{xx} \\ Z_x &= (XP - XM)/(2 * \Delta x) \\ Z_y &= (YP - YM)/(2 * \Delta y) \\ Z_{yy} &= (YP - 2Z + YM)/(\Delta y^2) \\ Z_{xx} &= (XP - 2Z + XM)/(\Delta x^2) \\ Z_{xy} &= (XYPP + XYMM - XYPM - XYMP)/(4\Delta x \Delta y) \\ Z &= Res * 8\Delta x^2 \Delta y^2 \end{aligned} \tag{B.7}$$

$$
\begin{aligned}
yMmP &= yM - yP \\
xMmP &= xM - xP \\
yMmP2Z &= yM + yP - 2 * Z \\
xyComb &= xyMM - xyMP - xyPM + xyPP \\
xMmP2 &= xMmP * xMmP \\
yMmP2 &= yMmP * yMmP \\
xyComb2 &= xMmP * yMmP \\
bot &= 1/(8 * \Delta x^2 * \Delta y^2)
\end{aligned}
\tag{B.8}
$$

$$
\begin{aligned}
dz &= -4 * (4 * (\Delta x^2 + \Delta y^2) + xMmP2 + yMmP2) \\
\Delta yMxM &= -xyComb2 \\
\Delta yPxM &= xyComb2 \\
\Delta yMxP &= xyComb2 \\
\Delta yPxP &= -xyComb2 \\
\Delta xM &= 8 * \Delta y^2 - (xyComb - 2 * yMmP) * yMmP \\
&\quad +4 * xMmP * yMmP2Z \\
\Delta xP &= 8 * \Delta y^2 + (xyComb + 2 * yMmP) * yMmP \\
&\quad -4 * xMmP * yMmP2Z \\
\Delta yP &= 8 * \Delta x^2 + 2 * xMmP2 \\
&\quad -xP * (xyComb + 4 * yMmP) \\
&\quad +xM * (xyComb - 4 * yMmP) \\
&\quad +8 * yMmP * Z \\
\Delta yM &= 8 * \Delta x^2 + 2 * xMmP2 \\
&\quad -xM * (xyComb - 4 * yMmP) \\
&\quad +xP * (xyComb + 4 * yMmP) \\
&\quad -8 * yMmP * Z \\
B0 &= -(2 * ((4 * \Delta y^2 + yMmP2) \\
&\quad *(xM + xP - 2 * Z) \\
&\quad +(4 * \Delta x^2 + xMmP2) * yMmP2Z) \\
&\quad -xyComb2 * xyComb)
\end{aligned}
\tag{B.9}
$$

B.3 Linearization D

$$
\begin{aligned}
P1 &= (1 + Z_x^2) * Z_{yy} \\
P2 &= 2 * Z_x * Z_y * Z_{xy} \\
P3 &= (1 + Z_y^2) * Z_{xx} \\
Z_x &= (XP - XM)/(2 * \Delta x) \\
Z_{yy} &= (-Y3M + 4 * Y2M - 5 * YM + 2 * Z)/(\Delta y^2) \\
Z_y &= (Y2M - 4 * YM + 3 * Z)/(2 * \Delta y) \\
Z_{xx} &= (XP - 2 * Z + XM)/(\Delta x^2) \\
Z_{xy} &= (((Y2MXP - Y2MXM)/(2 * \Delta x)) \\
&\quad -4 * ((YMXP - YMXM)/(2 * \Delta x)) + 3 * Z_x)/(2 * \Delta y) \\
Z &= Res * 8\Delta x^2 \Delta y^2
\end{aligned}
\tag{B.10}
$$

$$
\begin{aligned}
bot &= 1/(8 * \Delta x^2 * \Delta y^2) \\
xMmP &= xM - xP \\
yMmP &= y2M - 4 * yM + 3 * Z \\
xMP2Z &= xM + xP - 2 * Z \\
yComp1 &= 5 * y2M - 2 * y3M + 2 * yM - 5 * Z \\
xyComb &= y2MxM - y2MxP - 4 * yMxM + 4 * yMxP \\
xMmP2 &= xMmP * xMmP \\
yMmP2 &= yMmP * yMmP \\
xyComb1 &= xMmP * yMmP \\
xyComb2 &= 2 * xMmP * yComp1 - 2 * yMmP2 * xyComb \\
x2Comb &= 8 * \Delta x^2 + 2 * xMmP2 \\
y2Comb &= 8 * \Delta y^2 + 2 * yMmP2
\end{aligned}
\tag{B.11}
$$

$$
\begin{aligned}
\Delta y3M &= -x2Comb \\
\Delta y2MxP &= xyComb1 \\
\Delta yMxM &= 4 * xyComb1 \\
\Delta y2MxM &= -xyComb1 \\
\Delta yMxP &= -4 * xyComb1 \\
dz &= 2 * (x2Comb - y2Comb) \\
&\quad -xMmP2 * 9 * xyComb \\
&\quad +12 * xMP2Z * yMmP \\
\Delta xM &= 8 * \Delta y^2 + xyComb2 \\
\Delta xP &= 8 * \Delta y^2 - xyComb2 \\
\Delta yM &= -5 * x2Comb \\
&\quad +12 * xMmP2 * xyComb \\
&\quad -16 * xMP2Z * yMmP \\
\Delta y2M &= 4 * x2Comb \\
&\quad +3 * xMmP2 * xyComb \\
&\quad +4 * xMP2Z * yMmP \\
B0 &= -(y2Comb * xMP2Z \\
&\quad +x2Comb * (4 * y2M - y3M \\
&\quad -5 * yM + 2 * Z) \\
&\quad -3 * xMmP2 * xyComb * yMmP)
\end{aligned}
\tag{B.12}
$$

B.4 Linearization K

$$
\begin{aligned}
P1 &= (1 + Z_x^2) * Z_{yy} \\
P2 &= 2 * Z_x * Z_y * Z_{xy} \\
P3 &= (1 + Z_y^2) * Z_{xx} \\
Z_x &= (X2M - 8XM + 8XP - X2P)/(12 * \Delta x) \\
Z_{xx} &= (-X2M + 16XM - 30Z + 16XP - X2P)/(12 * \Delta x^2) \\
Z_y &= (Y2M - 4YM + 3Z)/(2 * \Delta y) \\
Z_{yy} &= (-Y3M + 4Y2M - 5YM + 2Z)/(\Delta y^2) \\
Z_{xy} &= (((Y2MX2M - 4YMX2M + 3X2M)/(2 * \Delta y)) \\
&\quad -8((Y2MXM - 4YMXM + 3XM)/(2 * \Delta y)) \\
&\quad +8((Y2MXP - 4YMXP + 3XP)/(2 * \Delta y)) \\
&\quad -((Y2MX2P - 4YMX2P + 3X2P)/(2 * \Delta y)))/(12 * \Delta x) \\
Z &= Res * 288 \Delta x^2 \Delta y^2
\end{aligned}
\tag{B.13}
$$

$$
\begin{aligned}
xMmP &= x2M - x2P - 8 * xM + 8 * xP \\
yMmP &= y2M - 4 * yM + 3 * z \\
yComp1 &= 4 * y2M - y3M - 5 * yM + 2 * z \\
Inner1 &= 3 * x2M - 3 * x2P - 24 * xM \\
&\quad +24 * xP + y2Mx2M - y2Mx2P - 8 * y2MxM \\
&\quad +8 * y2MxP - 4 * yMx2M + 4 * yMx2P \\
&\quad +32 * yMxM - 32 * yMxP \\
xComp1 &= x2M + x2P - 16 * (xM + xP) + 30 * z \\
xyComb &= xMmP * yMmP \\
Inner2 &= 4 * xMmP * yComp1 \\
&\quad -3 * xyComb - Inner1 * yMmP \\
Inner3 &= xMmP * Inner1 + 12 * yMmP * xComp1 \\
x2Comb &= 288 * \Delta x^2 + 2 * xMmP * xMmP \\
y2Comb &= 24 * \Delta y^2 + 6 * yMmP * yMmP \\
bot &= 1/(288 * \Delta x^2 * \Delta y^2)
\end{aligned}
\tag{B.14}
$$

$$
\begin{aligned}
\Delta y3M &= -x2Comb \\
\Delta y2Mx2M &= -xyComb \\
\Delta y2MxP &= -8 * xyComb \\
\Delta yMx2P &= -4 * xyComb \\
\Delta yMxM &= -32 * xyComb \\
\Delta y2Mx2P &= xyComb \\
\Delta y2MxM &= 8 * xyComb \\
\Delta yMx2M &= 4 * xyComb \\
\Delta yMxP &= 32 * xyComb \\
\Delta x2P &= -y2Comb - Inner2 \\
\Delta x2M &= -y2Comb + Inner2 \\
dz &= 2 * x2Comb - 30 * y2Comb - 3 * Inner3 \\
\Delta y2M &= 4 * x2Comb - Inner3 \\
\Delta xM &= 16 * y2Comb - 8 * Inner2 \\
\Delta xP &= 16 * y2Comb + 8 * Inner2 \\
\Delta yM &= -5 * x2Comb + 4 * Inner3 \\
B0 &= -x2Comb * yComp1 \\
&\quad +xyComb * Inner1 \\
&\quad +y2Comb * xComp1
\end{aligned}
\tag{B.15}
$$

B.5 Linearization B

$$
\begin{aligned}
P1 &= (1 + Z_x^2) * Z_{yy} \\
P2 &= 2 * Z_x * Z_y * Z_{xy} \\
P3 &= (1 + Z_y^2) * Z_{xx} \\
Z_x &= (X2M - 8XM + 8XP - X2P)/(12 * \Delta x) \\
Z_{xx} &= (-X2M + 16XM - 30Z + 16XP - X2P)/(12 * \Delta x^2) \\
Z_y &= (Y2M - 8YM + 8YP - Y2P)/(12 * \Delta y) \\
Z_{yy} &= (-Y2M + 16YM - 30Z + 16YP - Y2P)/(12 * \Delta y^2) \\
Z_{xy} &= (((Y2MX2M - 8Y2MXM + 8Y2MXP - Y2MX2P)/(12\Delta x)) \\
&\quad -8((YMX2M - 8YMXM + 8YMXP - YMX2P)/(12\Delta x)) \\
&\quad +8((YPX2M - 8YPXM + 8YPXP - YPX2P)/(12\Delta x)) \\
&\quad -((Y2PX2M - 8Y2PXM + 8Y2PXP - Y2PX2P)/(12\Delta x))) \\
&\quad /(12\Delta y) \\
Z &= Res * 10368\Delta x^2 \Delta y^2
\end{aligned}
\tag{B.16}
$$

$$
\begin{aligned}
bot &= 1/(10368 * \Delta x^2 * \Delta y^2) \\
xMmP &= x2M - x2P - 8 * xM + 8 * xP \\
yMmP &= y2M - y2P - 8 * yM + 8 * yP \\
yComb1 &= y2M + y2P - 16 * (yM + yP) + 30 * z \\
xComb1 &= x2M + x2P - 16 * (xM + xP) + 30 * z \\
Inner1 &= y2Mx2M - y2Mx2P \\
&\quad -8 * (y2MxM - y2MxP) - y2Px2M + y2Px2P \\
&\quad +8 * (y2PxM - y2PxP - yMx2M + yMx2P \\
&\quad +8 * (yMxM - yMxP) + yPx2M - yPx2P \\
&\quad -8 * yPxM + 8 * yPxP) \\
xyComb &= xMmP * yMmP \\
xInner &= xMmP * Inner1 + 12 * yMmP * xComb1 \\
yInner &= yMmP * Inner1 + 12 * xMmP * yComb1 \\
xMmP2 &= 6 * xMmP * xMmP \\
yMmP2 &= 6 * yMmP * yMmP \\
x2Comp &= 864 * \Delta x^2 + xMmP2 \\
y2Comp &= 864 * \Delta y^2 + yMmP2
\end{aligned}
\tag{B.17}
$$

$$
\begin{aligned}
\Delta y2Mx2M &= -xyComb \\
\Delta y2MxP &= -8 * xyComb \\
\Delta y2Px2P &= -xyComb \\
\Delta y2PxM &= -8 * xyComb \\
\Delta yMx2P &= -8 * xyComb \\
\Delta yMxM &= -64 * xyComb \\
\Delta yPx2M &= -8 * xyComb \\
\Delta yPxP &= -64 * xyComb \\
\Delta y2Mx2P &= xyComb \\
\Delta y2MxM &= 8 * xyComb \\
\Delta y2Px2M &= xyComb \\
\Delta y2PxP &= 8 * xyComb \\
\Delta yMx2M &= 8 * xyComb \\
\Delta yMxP &= 64 * xyComb \\
\Delta yPx2P &= 8 * xyComb \\
\Delta yPxM &= 64 * xyComb \\
dz &= -30 * (x2Comp + y2Comp) \\
\Delta y2M &= -x2Comp - xInner \\
\Delta y2P &= -x2Comp + xInner \\
\Delta x2M &= -y2Comp - yInner \\
\Delta x2P &= -y2Comp + yInner \\
\Delta yP &= 16 * x2Comp - 8 * xInner \\
\Delta yM &= 16 * x2Comp + 8 * xInner \\
\Delta xP &= 16 * y2Comp - 8 * yInner \\
\Delta xM &= 16 * y2Comp + 8 * yInner \\
B0 &= y2Comp * xComb1 \\
&\quad + x2Comp * yComb1 \\
&\quad + xyComb * Inner1
\end{aligned}
\tag{B.18}
$$

B.6 Linearization L

$$
\begin{aligned}
P1 &= (1+Z_x^2)*Z_{yy} \\
P2 &= 2*Z_x*Z_y*Z_{xy} \\
P3 &= (1+Z_y^2)*Z_{xx} \\
Z_x &= (X2M-8XM+8XP-X2P)/(12*\Delta x) \\
Z_{xx} &= (-X2M+16XM-30Z+16XP-X2P)/(12*\Delta x^2) \\
Z_y &= (-3YM-Z+5YP-Y2P)/(6*\Delta y) \\
Z_{yy} &= (11YM-20Z+6YP+4Y2P-Y3P)/(11*\Delta y^2) \\
Z_{xy} &= (((-3X2M+4YPX2M-Y2PX2M)/(2\Delta y)) \\
&\quad -8((-3XM+4YPXM-Y2PXM)/(2\Delta y)) \\
&\quad +8((-3XP+4YPXP-Y2PXP)/(2\Delta y)) \\
&\quad -((-3X2P+4YPX2P-Y2PX2P)/(2\Delta y)))/(12*\Delta x) \\
Z &= Res*9504\Delta x^2\Delta y^2
\end{aligned}
\tag{B.19}
$$

$$
\begin{aligned}
xMmP &= x2M - x2P - 8 * xM + 8 * xP \\
yMmP &= y2P + 3 * yM - 5 * yP + z \\
yComb1 &= 4 * y2P - y3P + 11 * yM + 6 * yP - 20 * z \\
xComb1 &= x2M + x2P - 16 * (xM + xP) + 30 * z \\
Inner1 &= 3 * x2M - 3 * x2P - 24 * xM + 24 * xP \\
&\quad + y2Px2M - y2Px2P - 8 * y2PxM + 8 * y2PxP \\
&\quad - 4 * yPx2M + 4 * yPx2P + 32 * yPxM - 32 * yPxP \\
xyComb &= xMmP * yMmP \\
Inner2 &= 12 * xMmP * yComb1 - 33 * xyComb \\
&\quad - 11 * Inner1 * yMmP \\
xMmP2 &= xMmP * xMmP \\
yMmP2 &= yMmP * yMmP \\
Inner3 &= -xMmP * Inner1 - 4 * yMmP * xComb1 \\
x2Comb &= 864 * \Delta x^2 + 6 * xMmP2 \\
y2Comb &= 792 * \Delta y^2 + 22 * yMmP2 \\
bot &= 1/(9504 * \Delta x^2 * \Delta y^2)
\end{aligned}
\tag{B.20}
$$

$$
\begin{aligned}
\Delta y3P &= -x2Comb \\
\Delta y2Px2M &= -11 * xyComb \\
\Delta y2Px2P &= 11 * xyComb \\
\Delta y2PxM &= 88 * xyComb \\
\Delta y2PxP &= -88 * xyComb \\
\Delta yPx2M &= 44 * xyComb \\
\Delta yPx2P &= -44 * xyComb \\
\Delta yPxM &= -352 * xyComb \\
\Delta yPxP &= 352 * xyComb \\
\Delta x2M &= -y2Comb + Inner2 \\
\Delta x2P &= -y2Comb - Inner2 \\
\Delta y2P &= 4 * x2Comb + 11 * Inner3 \\
dz &= -20 * x2Comb - 30 * y2Comb + 11 * Inner3 \\
\Delta yP &= 6 * x2Comb - 5 * Inner3 \\
\Delta xP &= 16 * y2Comb + 8 * Inner2 \\
\Delta xM &= 16 * y2Comb - 8 * Inner2 \\
\Delta yM &= 11 * x2Comb + 33 * Inner3 \\
B0 &= -x2Comb * yComb1 + y2Comb * xComb1 \\
&\quad +11 * xyComb * Inner1
\end{aligned}
\tag{B.21}
$$

B.7 Linearization N

$$
\begin{aligned}
P1 &= (1+Z_x^2)*Z_{yy} \\
P2 &= 2*Z_x*Z_y*Z_{xy} \\
P3 &= (1+Z_y^2)*Z_{xx} \\
Z_x &= (-3XM - Z + 5XP - X2P)/(6*\Delta x) \\
Z_{xx} &= (11XM - 20Z + 6XP + 4X2P - X3P)/(11*\Delta x^2) \\
Z_y &= (Y2M - 8YM + 8YP - Y2P)/(12*\Delta y) \\
Z_{yy} &= (-Y2M + 16YM - 30Z + 16YP - Y2P)/(12*\Delta y^2) \\
Z_{xy} &= (((-3Y2M + 4Y2MXP - Y2MX2P)/(2\Delta x)) \\
&\quad -8((-3YM + 4YMXP - YMX2P)/(2\Delta x)) \\
&\quad +8((-3YP + 4YPXP - YPX2P)/(2\Delta x)) \\
&\quad -((-3Y2P + 4Y2PXP - Y2PX2P)/(2\Delta x)))/(12\Delta y) \\
Z &= Res*9504\Delta x^2\Delta y^2
\end{aligned}
\tag{B.22}
$$

$$
\begin{aligned}
xMmP &= x2P + 3 * xM - 5 * xP + z \\
yMmP &= y2M - y2P - 8 * yM + 8 * yP \\
xComb1 &= 4 * x2P - x3P + 11 * xM + 6 * xP - 20 * z \\
yComb1 &= y2M + y2P - 16 * (yM + yP) + 30 * z \\
Inner1 &= 3 * y2M + y2Mx2P - 4 * y2MxP \\
&\quad -3 * y2P - y2Px2P + 4 * (y2PxP - 6 * yM \\
&\quad -2 * yMx2P + 8 * yMxP + 6 * yP \\
&\quad +2 * yPx2P - 8 * yPxP) \\
xyComb &= yMmP * xMmP \\
yMmP2 &= yMmP * yMmP \\
xMmP2 &= xMmP * xMmP \\
Inner2 &= 12 * yMmP * xComb1 - 33 * xyComb \\
&\quad -11 * Inner1 * xMmP \\
xyComb2 &= xMmP * yComb1 \\
xyComb4 &= yMmP * Inner1 + 4 * xyComb2 \\
x2Comb &= 792 * \Delta x^2 + 22 * xMmP2 \\
y2Comb &= 864 * \Delta y^2 + 6 * yMmP2 \\
bot &= 1/(9504 * \Delta x^2 * \Delta y^2)
\end{aligned}
\tag{B.23}
$$

$$
\begin{aligned}
\Delta x3P &= -y2Comb \\
\Delta y2Mx2P &= -11 * xyComb \\
\Delta y2MxP &= 44 * xyComb \\
\Delta y2Px2P &= 11 * xyComb \\
\Delta y2PxP &= -44 * xyComb \\
\Delta yMx2P &= 88 * xyComb \\
\Delta yMxP &= -352 * xyComb \\
\Delta yPx2P &= -88 * xyComb \\
\Delta yPxP &= 352 * xyComb \\
\Delta y2M &= -x2Comb + Inner2 \\
\Delta y2P &= -x2Comb - Inner2 \\
\Delta x2P &= 4 * y2Comb - 11 * xyComb4 \\
dz &= -30 * x2Comb - 20 * y2Comb - 11 * xyComb4 \\
\Delta xP &= 6 * y2Comb + 55 * xyComb4 \\
\Delta yP &= 16 * x2Comb + 8 * Inner2 \\
\Delta yM &= 16 * x2Comb - 8 * Inner2 \\
\Delta xM &= 11 * y2Comb - 33 * xyComb4 \\
B0 &= -y2Comb * xComb1 + 11 * xyComb * Inner1 \\
&\quad +x2Comb * yComb1
\end{aligned}
\tag{B.24}
$$

B.8 Linearization Q

$$
\begin{aligned}
P1 &= (1+Z_x^2)*Z_{yy} \\
P2 &= 2*Z_x*Z_y*Z_{xy} \\
P3 &= (1+Z_y^2)*Z_{xx} \\
Z_x &= (X2M-5XM+Z+3XP)/(6*\Delta x) \\
Z_{xx} &= (-X3M+4X2M+6XM-20Z+11XP)/(11*\Delta x^2) \\
Z_y &= (Y2M-8YM+8YP-Y2P)/(12*\Delta y) \\
Z_{yy} &= (-Y2M+16YM-30Z+16YP-Y2P)/(12*\Delta y^2) \\
Z_{xy} &= (((Y2MX2M-4Y2MXM+3Y2M)/(2\Delta x)) \\
&\quad -8((YMX2M-4YMXM+3YM)/(2\Delta x)) \\
&\quad +8((YPX2M-4YPXM+3YP)/(2\Delta x)) \\
&\quad -((Y2PX2M-4Y2PXM+3Y2P)/(2\Delta x)))/(12*\Delta y) \\
Z &= Res*9504\Delta x^2\Delta y^2
\end{aligned}
\tag{B.25}
$$

$$
\begin{aligned}
xMmP &= x2M - 5 * xM + 3 * xP + z \\
yMmP &= y2M - y2P - 8 * yM + 8 * yP \\
xComb1 &= 4 * x2M - x3M + 6 * xM + 11 * xP - 20 * z \\
yComb1 &= y2M + y2P - 16 * (yM + yP) + 30 * z \\
Inner1 &= 3 * y2M + y2Mx2M - 3 * y2P - y2Px2M \\
&\quad +4 * (y2PxM - 6 * yM - 2 * yMx2M + 8 * yMxM \\
&\quad +6 * yP + 2 * yPx2M - 8 * yPxM - y2MxM) \\
xyComb &= yMmP * xMmP \\
Inner2 &= 12 * yMmP * xComb1 \\
&\quad -33 * xyComb - 11 * Inner1 * xMmP \\
Inner3 &= yMmP * Inner1 + 4 * xMmP * yComb1 \\
yMmP2 &= yMmP * yMmP \\
xMmP2 &= xMmP * xMmP \\
x2Comb &= 792 * \Delta x^2 + 22 * xMmP2 \\
y2Comb &= 864 * \Delta y^2 + 6 * yMmP2 \\
bot &= 1/(9504 * \Delta x^2 * \Delta y^2)
\end{aligned}
\tag{B.26}
$$

$$
\begin{aligned}
\Delta x3M &= -y2Comb \\
\Delta y2Mx2M &= -11 * xyComb \\
\Delta y2PxM &= -44 * xyComb \\
\Delta yMxM &= -352 * xyComb \\
\Delta yPx2M &= -88 * xyComb \\
\Delta y2MxM &= 44 * xyComb \\
\Delta y2Px2M &= 11 * xyComb \\
\Delta yMx2M &= 88 * xyComb \\
\Delta yPxM &= 352 * xyComb \\
\Delta y2M &= -x2Comb + Inner2 \\
\Delta y2P &= -x2Comb - Inner2 \\
\Delta x2M &= 4 * y2Comb - 11 * Inner3 \\
dz &= -30 * x2Comb - 20 * y2Comb - 11 * Inner3 \\
\Delta xM &= 6 * y2Comb + 55 * Inner3 \\
\Delta yP &= 16 * x2Comb + 8 * Inner2 \\
\Delta yM &= 16 * x2Comb - 8 * Inner2 \\
\Delta xP &= 11 * y2Comb - 33 * Inner3 \\
B0 &= -y2Comb * xComb1 \\
&\quad +11 * xyComb * Inner1 + x2Comb * yComb1
\end{aligned}
\tag{B.27}
$$

B.9 Linearization M

$$
\begin{aligned}
P1 &= (1+Z_x^2)*Z_{yy} \\
P2 &= 2*Z_x*Z_y*Z_{xy} \\
P3 &= (1+Z_y^2)*Z_{xx} \\
Z_x &= (-3XM - Z + 5XP - X2P)/(6*\Delta x) \\
Z_{xx} &= (11XM - 20Z + 6XP + 4X2P - X3P)/(11*\Delta x^2) \\
Z_y &= (-3YM - Z + 5YP - Y2P)/(6*\Delta y) \\
Z_{yy} &= (11YM - 20Z + 6YP + 4Y2P - Y3P)/(11*\Delta y^2) \\
Z_{xy} &= (-3((-3XM + 4YPXM - Y2PXM)/(2\Delta y)) \\
&\quad -((-3Z + 4YP - Y2P)/(2\Delta y)) \\
&\quad +5((-3XP + 4YPXP - Y2PXP)/(2\Delta y)) \\
&\quad -((-3X2P + 4YPX2P - Y2PX2P)/(2\Delta y)))/(6*\Delta x) \\
Z &= Res*2376\Delta x^2\Delta y^2
\end{aligned}
\tag{B.28}
$$

$$
\begin{aligned}
xMmP &= x2P + 3 * xM - 5 * xP + z \\
yMmP &= y2P + 3 * yM - 5 * yP + z \\
xComb1 &= 4 * x2P - x3P + 11 * xM \\
&\quad +6 * xP - 20 * z \\
yComb1 &= 4 * y2P - y3P + 11 * yM \\
&\quad +6 * yP - 20 * z \\
Inner1 &= 3 * x2P + 9 * xM - 15 * xP \\
&\quad +y2P + y2Px2P + 3 * y2PxM - 5 * y2PxP \\
&\quad -4 * (yP + yPx2P + 3 * yPxM - 5 * yPxP) \\
&\quad +3 * z \\
xMmP2 &= xMmP * xMmP \\
yMmP2 &= yMmP * yMmP \\
xyComb &= xMmP * yMmP \\
Inner2 &= 12 * yComb1 * xMmP - 33 * xyComb \\
&\quad -11 * yMmP * Inner1 \\
xComb2 &= 11 * xMmP * Inner1 \\
yComb2 &= 12 * xComb1 * yMmP \\
x2Comb &= 216 * \Delta x^2 + 6 * xMmP2 \\
y2Comb &= 216 * \Delta y^2 + 6 * yMmP2 \\
xyComb2 &= yComb2 - xComb2 \\
bot &= 1/(2376 * \Delta x^2 * \Delta y^2)
\end{aligned}
\tag{B.29}
$$

$$
\begin{aligned}
\Delta y2Px2P &= -11 * xyComb \\
\Delta y2PxM &= -33 * xyComb \\
\Delta y2PxP &= 55 * xyComb \\
\Delta yPx2P &= 44 * xyComb \\
\Delta yPxM &= 132 * xyComb \\
\Delta yPxP &= -220 * xyComb \\
\Delta y3P &= -x2Comb \\
\Delta x3P &= -y2Comb \\
\Delta y2P &= 4 * x2Comb + xyComb2 \\
&\quad -11 * xyComb \\
\Delta yP &= 6 * x2Comb - 5 * xyComb2 \\
&\quad +44 * xyComb \\
\Delta x2P &= 4 * y2Comb + Inner2 \\
dz &= -20 * (x2Comb + y2Comb) \\
&\quad +xyComb2 + Inner2 \\
\Delta xP &= 6 * y2Comb - 5 * Inner2 \\
\Delta yM &= 11 * x2Comb + 3 * xyComb2 \\
\Delta xM &= 11 * y2Comb + 3 * Inner2 \\
B0 &= -y2Comb * xComb1 - x2Comb * yComb1 \\
&\quad +11 * xyComb * Inner1
\end{aligned}
\tag{B.30}
$$

B.10 Linearization P

$$
\begin{aligned}
P1 &= (1 + Z_x^2) * Z_{yy} \\
P2 &= 2 * Z_x * Z_y * Z_{xy} \\
P3 &= (1 + Z_y^2) * Z_{xx} \\
Z_x &= (X2M - 5XM + Z + 3XP)/(6 * \Delta x) \\
Z_{xx} &= (-X3M + 4X2M + 6XM - 20Z + 11XP)/(11 * \Delta x^2) \\
Z_y &= (-3YM - Z + 5YP - Y2P)/(6 * \Delta y) \\
Z_{yy} &= (11YM - 20Z + 6YP + 4Y2P - Y3P)/(11 * \Delta y^2) \\
Z_{xy} &= (-3((YMX2M - 4YMXM + 3YM)/(2\Delta x)) \\
&\quad -((X2M - 4XM + 3Z)/(2\Delta x)) \\
&\quad +5((YPX2M - 4YPXM + 3YP)/(2\Delta x)) \\
&\quad -((Y2PX2M - 4Y2PXM + 3Y2P)/(2\Delta x)))/(6\Delta y) \\
Z &= Res * 2376\Delta x^2 \Delta y^2
\end{aligned}
\tag{B.31}
$$

$$
\begin{aligned}
xMmP &= x2M - 5 * xM + 3 * xP + z \\
yMmP &= y2P + 3 * yM - 5 * yP + z \\
xComb1 &= 4 * x2M - x3M + 6 * xM + 11 * xP - 20 * z \\
yComb1 &= 4 * y2P - y3P + 11 * yM + 6 * yP - 20 * z \\
Inner1 &= x2M - 4 * xM + 3 * y2P + y2Px2M \\
&\quad -4 * y2PxM + 9 * yM + 3 * yMx2M - 12 * yMxM \\
&\quad -5 * (3 * yP + yPx2M - 4 * yPxM) + 3 * z \\
xyComb1 &= xMmP * yMmP \\
xMmP2 &= xMmP * xMmP \\
yMmP2 &= yMmP * yMmP \\
xyComb2 &= 12 * xComb1 * yMmP - 11 * xMmP * Inner1 \\
yComb4 &= 12 * yComb1 * xMmP - 11 * yMmP * Inner1 \\
x2Comb &= 216 * \Delta x^2 + 6 * xMmP2 \\
y2Comb &= 216 * \Delta y^2 + 6 * yMmP2 \\
bot &= 1/(2376 * \Delta x^2 * \Delta y^2)
\end{aligned}
\tag{B.32}
$$

$$
\begin{aligned}
\Delta y2Px2M &= -11 * xyComb1 \\
\Delta y2PxM &= 44 * xyComb1 \\
\Delta yMx2M &= -33 * xyComb1 \\
\Delta yMxM &= 132 * xyComb1 \\
\Delta yPx2M &= 55 * xyComb1 \\
\Delta yPxM &= -220 * xyComb1 \\
\Delta y3P &= -x2Comb \\
\Delta x3M &= -y2Comb \\
\Delta y2P &= 4 * x2Comb + xyComb2 - 33 * xyComb1 \\
\Delta yP &= 6 * x2Comb - 5 * xyComb2 + 165 * xyComb1 \\
\Delta x2M &= 4 * y2Comb + yComb4 - 11 * xyComb1 \\
dz &= -20 * (x2Comb + y2Comb) + yComb4 \\
&\quad + xyComb2 - 33 * xyComb1 \\
\Delta xM &= 6 * y2Comb - 5 * yComb4 + 44 * xyComb1 \\
\Delta yM &= 11 * x2Comb + 3 * (xyComb2 - 33 * xyComb1) \\
\Delta xP &= 11 * y2Comb + 3 * yComb4 \\
B0 &= -y2Comb * xComb1 - x2Comb * yComb1 \\
&\quad + 11 * xyComb1 * Inner1
\end{aligned}
\tag{B.33}
$$

B.11 Linearization U

$$
\begin{aligned}
P1 &= (1 + Z_x^2) * Z_{yy} \\
P2 &= 2 * Z_x * Z_y * Z_{xy} \\
P3 &= (1 + Z_y^2) * Z_{xx} \\
Z_x &= (X2M - 5XM + Z + 3XP)/(6 * \Delta x) \\
Z_{xx} &= (-X3M + 4X2M + 6XM - 20Z + 11XP)/(11 * \Delta x^2) \\
Z_y &= (Y2M - 5YM + Z + 3YP)/(6 * \Delta y) \\
Z_{yy} &= (-Y3M + 4Y2M + 6YM - 20Z + 11YP)/(11 * \Delta y^2) \\
Z_{xy} &= (((Y2MX2M - 4Y2MXM + 3Y2M)/(2\Delta x)) \\
&\quad -5((YMX2M - 4YMXM + 3YM)/(2\Delta x)) \\
&\quad +((X2M - 4XM + 3Z)/(2\Delta x)) \\
&\quad +3((YPX2M - 4YPXM + 3YP)/(2\Delta x)))/(6\Delta y) \\
Z &= Res * 2376\Delta x^2 \Delta y^2
\end{aligned}
\tag{B.34}
$$

$$
\begin{aligned}
xMmP &= x2M - 5 * xM + 3 * xP + z \\
yMmP &= y2M - 5 * yM + 3 * yP + z \\
Inner1 &= x2M - 4 * xM + 3 * y2M \\
&\quad + y2Mx2M - 4 * y2MxM - 15 * yM \\
&\quad - 5 * yMx2M + 20 * yMxM \\
&\quad + 3 * (3 * yP + yPx2M - 4 * yPxM + z) \\
xComb1 &= 4 * x2M - x3M + 6 * xM \\
&\quad + 11 * xP - 20 * z \\
yComb1 &= 4 * y2M - y3M + 6 * yM \\
&\quad + 11 * yP - 20 * z \\
xyComb1 &= xMmP * yMmP \\
x2Comb &= 216 * \Delta x^2 + 6 * xMmP * xMmP \\
y2Comb &= 216 * \Delta y^2 + 6 * yMmP * yMmP \\
xyComb3 &= 12 * yComb1 * xMmP \\
&\quad - 11 * yMmP * Inner1 \\
Inner2 &= 12 * xComb1 * yMmP - 33 * xyComb1 \\
&\quad - 11 * xMmP * Inner1 \\
bot &= 1/(2376 * \Delta x^2 * \Delta y^2)
\end{aligned}
\tag{B.35}
$$

$$
\begin{aligned}
B0 &= -y2Comb * xComb1 - x2Comb * yComb1 \\
&\quad +11 * xyComb1 * Inner1 \\
\Delta y2Mx2M &= -11 * xyComb1 \\
\Delta y2MxM &= 44 * xyComb1 \\
\Delta yMx2M &= 55 * xyComb1 \\
\Delta yMxM &= -220 * xyComb1 \\
\Delta yPx2M &= -33 * xyComb1 \\
\Delta yPxM &= 132 * xyComb1 \\
\Delta y3M &= -x2Comb \\
\Delta x3M &= -x2Comb \\
\Delta y2M &= 4 * x2Comb + Inner2 \\
\Delta yM &= 6 * x2Comb - 5 * Inner2 \\
\Delta x2M &= 4 * y2Comb + xyComb3 - 11 * xyComb1 \\
dz &= -20 * (x2Comb + y2Comb) + xyComb3 + Inner2 \\
\Delta xM &= 6 * y2Comb - 5 * xyComb3 + 44 * xyComb1 \\
\Delta yP &= 11 * x2Comb + 3 * Inner2 \\
\Delta xP &= 11 * y2Comb + 3 * xyComb3
\end{aligned}
\tag{B.36}
$$

B.12 Linearization S

$$
\begin{aligned}
P1 &= (1 + Z_x^2) * Z_{yy} \\
P2 &= 2 * Z_x * Z_y * Z_{xy} \\
P3 &= (1 + Z_y^2) * Z_{xx} \\
Z_x &= (X2M - 5XM + Z + 3XP)/(6 * \Delta x) \\
Z_{xx} &= (-X3M + 4X2M + 6XM - 20Z + 11XP)/(11 * \Delta x^2) \\
Z_y &= (YP - YM)/(2 * \Delta y) \\
Z_{yy} &= (YP - 2Z + YM)/(\Delta y^2) \\
Z_{xy} &= (((YPX2M - YMX2M)/(2\Delta y)) \\
&\quad -4((YPXM - YMXM)/(2\Delta y)) \\
&\quad +3((YP - YM)/(2\Delta y)))/(6\Delta x) \\
Z &= Res * 792\Delta x^2 \Delta y^2
\end{aligned}
\tag{B.37}
$$

$$
\begin{aligned}
xMmP &= x2M - 5 * xM + 3 * xP + z \\
yMmP &= yM - yP \\
xComb1 &= 4 * x2M - x3M + 6 * xM + 11 * xP - 20 * z \\
yComb1 &= 3 * yM + yMx2M - 4 * yMxM \\
&\quad -3 * yP - yPx2M + 4 * yPxM \\
yComb2 &= yM + yP - 2 * z \\
xyComb1 &= yMmP * xMmP \\
yComb3 &= -11 * yMmP * yComb1 \\
&\quad +44 * yComb2 * xMmP \\
xyComb2 &= 36 * yMmP * xComb1 - 33 * xyComb1 \\
&\quad -11 * yComb1 * xMmP \\
x2Comb &= 792 * \Delta x^2 + 22 * xMmP * xMmP \\
y2Comb &= 72 * \Delta y^2 + 18 * yMmP * yMmP \\
bot &= 1/(792 * \Delta x^2 * \Delta y^2)
\end{aligned}
\tag{B.38}
$$

$$
\begin{aligned}
\Delta x3M &= -y2Comb \\
\Delta yMxM &= 44 * xyComb1 \\
\Delta yPx2M &= 11 * xyComb1 \\
\Delta yMx2M &= -11 * xyComb1 \\
\Delta yPxM &= -44 * xyComb1 \\
dz &= -2 * x2Comb - 20 * y2Comb + yComb3 \\
\Delta yM &= x2Comb + xyComb2 \\
\Delta yP &= x2Comb - xyComb2 \\
\Delta xM &= 6 * y2Comb - 5 * yComb3 \\
\Delta x2M &= 4 * y2Comb + yComb3 \\
\Delta xP &= 11 * y2Comb - 3 * yComb3 \\
B0 &= -y2Comb * xComb1 - x2Comb * yComb2 \\
&\quad +11 * xyComb1 * yComb1
\end{aligned}
\tag{B.39}
$$

B.13 Linearization T

$$
\begin{aligned}
P1 &= (1 + Z_x^2) * Z_{yy} \\
P2 &= 2 * Z_x * Z_y * Z_{xy} \\
P3 &= (1 + Z_y^2) * Z_{xx} \\
Z_x &= (-3XM - Z + 5XP - X2P)/(6 * \Delta x) \\
Z_{xx} &= (11XM - 20Z + 6XP + 4X2P - X3P)/(11 * \Delta x^2) \\
Z_y &= (Y2M - 5YM + Z + 3YP)/(6 * \Delta y) \\
Z_{yy} &= (-Y3M + 4Y2M + 6YM - 20Z + 11YP)/(11 * \Delta y^2) \\
Z_{xy} &= (-3((Y2MXM - 4YMXM + 3XM)/(2\Delta y)) \\
&\quad -((Y2M - 4YM + 3Z)/(2\Delta y)) \\
&\quad +5((Y2MXP - 4YMXP + 3XP)/(2\Delta y)) \\
&\quad -((Y2MX2P - 4YMX2P + 3X2P)/(2\Delta y)))/(6\Delta x) \\
Z &= Res * 2376\Delta x^2 \Delta y^2
\end{aligned}
\tag{B.40}
$$

$$
\begin{aligned}
xMmP &= x2P + 3 * xM - 5 * xP + z \\
yMmP &= y2M - 5 * yM + 3 * yP + z \\
xComb1 &= 4 * x2P - x3P + 11 * xM + 6 * xP - 20 * z \\
yComb1 &= 4 * y2M - y3M + 6 * yM + 11 * yP - 20 * z \\
Inner1 &= 3 * x2P + 9 * xM - 15 * xP + y2M + y2Mx2P \\
&\quad +3 * y2MxM - 5 * y2MxP \\
&\quad -4 * (yM + yMx2P + 3 * yMxM - 5 * yMxP) + 3 * z \\
xyComb1 &= xMmP * yMmP \\
x2Comb &= 216 * \Delta x^2 + 6 * xMmP * xMmP \\
y2Comb &= 216 * \Delta y^2 + 6 * yMmP * yMmP \\
Inner2 &= 12 * yComb1 * xMmP - 33 * xyComb1 \\
&\quad -11 * yMmP * Inner1 \\
Inner3 &= 12 * xComb1 * yMmP - 11 * xMmP * Inner1 \\
bot &= 1/(2376 * \Delta x^2 * \Delta y^2)
\end{aligned}
\tag{B.41}
$$

$$
\begin{aligned}
\Delta y2Mx2P &= -11 * xyComb1 \\
\Delta y2MxM &= -33 * xyComb1 \\
\Delta y2MxP &= 55 * xyComb1 \\
\Delta yMx2P &= 44 * xyComb1 \\
\Delta yMxM &= 132 * xyComb1 \\
\Delta yMxP &= -220 * xyComb1 \\
\Delta y3M &= -y2Comb \\
\Delta x3P &= -y2Comb \\
\Delta y2M &= 4 * x2Comb + Inner3 - 11 * xyComb1 \\
\Delta yM &= 6 * x2Comb - 5 * Inner3 + 44 * xyComb1 \\
\Delta x2P &= 4 * y2Comb + Inner2 \\
dz &= -20 * (x2Comb + y2Comb) + Inner2 + Inner3 \\
\Delta xP &= 6 * y2Comb - 5 * Inner2 \\
\Delta yP &= 11 * x2Comb + 3 * Inner3 \\
\Delta xM &= 11 * y2Comb + 3 * Inner2 \\
B0 &= -y2Comb * xComb1 - x2Comb * yComb1 \\
&\quad +11 * xyComb1 * Inner1
\end{aligned}
\tag{B.42}
$$

B.14 Linearization R

$$
\begin{aligned}
P1 &= (1+Z_x^2)*Z_{yy} \\
P2 &= 2*Z_x*Z_y*Z_{xy} \\
P3 &= (1+Z_y^2)*Z_{xx} \\
Z_x &= (-3XM - Z + 5XP - X2P)/(6*\Delta x) \\
Z_{xx} &= (11XM - 20Z + 6XP + 4X2P - X3P)/(11*\Delta x^2) \\
Z_y &= (YP - YM)/(2*\Delta y) \\
Z_{yy} &= (YP - 2Z + YM)/(\Delta y^2) \\
Z_{xy} &= (-3Zy + 4((YPXP - YMXP)/(2\Delta y)) \\
&\quad -((YPX2P - YMX2P)/(2\Delta y)))/(2\Delta x) \\
Z &= Res*792\Delta x^2\Delta y^2
\end{aligned}
\tag{B.43}
$$

$$
\begin{aligned}
xMmP &= x2P + 3*xM - 5*xP + z \\
yMmP &= yM - yP \\
xComb1 &= 4*x2P - x3P + 11*xM + 6*xP - 20*z \\
yComb1 &= yM + yP - 2*z \\
Inner1 &= 3*yM + yMx2P - 4*yMxP - 3*yP \\
&\quad -yPx2P + 4*yPxP \\
xyComb1 &= yMmP*xMmP \\
x2Comb &= 792*\Delta x^2 + 22*xMmP*xMmP \\
y2Comb &= 72*\Delta y^2 + 18*yMmP*yMmP \\
Inner2 &= 36*yMmP*xComb1 - 99*xyComb1 \\
&\quad -33*Inner1*xMmP \\
Inner3 &= 33*yMmP*Inner1 - 44*yComb1*xMmP \\
bot &= 1/(792*\Delta x^2*\Delta y^2)
\end{aligned}
\tag{B.44}
$$

$$
\begin{aligned}
\Delta x3P &= -y2Comb \\
\Delta yMx2P &= -33.0 * xyComb1 \\
\Delta yMxP &= 132.0 * xyComb1 \\
\Delta yPx2P &= 33.0 * xyComb1 \\
\Delta yPxP &= -132.0 * xyComb1 \\
dz &= -2.0 * x2Comb - 20.0 * y2Comb - Inner3 \\
\Delta yM &= x2Comb + Inner2 \\
\Delta yP &= x2Comb - Inner2 \\
\Delta xP &= 6.0 * y2Comb + 5.0 * Inner3 \\
\Delta x2P &= 4.0 * y2Comb - Inner3 \\
\Delta xM &= 11.0 * y2Comb - 3.0 * Inner3 \\
B0 &= y2Comb * xComb1 + x2Comb * yComb1 \\
&\quad -33.0 * xyComb1 * Inner1
\end{aligned}
\tag{B.45}
$$

B.15 Linearization F

$$
\begin{aligned}
P1 &= (1 + Z_x^2) * Z_{yy} \\
P2 &= 2 * Z_x * Z_y * Z_{xy} \\
P3 &= (1 + Z_y^2) * Z_{xx} \\
Z_x &= (X2M - 4XM + 3Z)/(2\Delta x) \\
Z_{xx} &= (-X3M + 4X2M - 5XM + 2Z)/(\Delta x^2) \\
Z_y &= (-3Z + 4YP - Y2P)/(2\Delta y) \\
Z_{yy} &= (2Z - 5YP + 4Y2P - Y3P)/(\Delta y^2) \\
Z_{xy} &= (-3((X2M - 4XM + 3Z)/(2\Delta x)) \\
&\quad +4((YPX2M - 4YPXM + 3YP)/(2\Delta x)) \\
&\quad -((Y2PX2M - 4Y2PXM + 3Y2P)/(2\Delta x)))/(2\Delta y) \\
Z &= Res * 8\Delta x^2 \Delta y^2
\end{aligned}
\tag{B.46}
$$

$$
\begin{aligned}
xMmP &= x2M - 4 * xM + 3 * z \\
yMmP &= y2P - 4 * yP + 3 * z \\
yComb1 &= 4 * y2P - y3P - 5 * yP + 2 * z \\
xComb1 &= 4 * x2M - x3M - 5 * xM + 2 * z \\
Inner1 &= 3 * x2M - 12 * xM + 3 * y2P + y2Px2M \\
&\quad -4 * (y2PxM + 3 * yP + yPx2M - 4 * yPxM) \\
&\quad +9 * z \\
xyComb1 &= xMmP * yMmP \\
xMmP2 &= xMmP * xMmP \\
yMmP2 &= yMmP * yMmP \\
x2Comb &= 8 * \Delta x^2 + 2 * xMmP2 \\
y2Comb &= 8 * \Delta y^2 + 2 * yMmP2 \\
bot &= 1/(8 * \Delta x^2 * \Delta y^2)
\end{aligned}
\tag{B.47}
$$

$$
\begin{aligned}
\Delta y2Px2M &= -xyComb1 \\
\Delta y2PxM &= 4 * xyComb1 \\
\Delta yPx2M &= 4 * xyComb1 \\
\Delta yPxM &= -16 * xyComb1 \\
\Delta x2M &= 4 * y2Comb - 3 * yMmP2 + y2Px2M(-3 + 4 * yP) \\
&\quad -16 * yP * (y2PxM + yPx2M - 4 * yPxM) \\
&\quad +y2P(69 * +10 * x2M - 40 * xM - y2Px2M \\
&\quad +4 * y2PxM + 4 * yPx2M - 16 * yPxM - 30 * z) \\
&\quad +12 * (y2PxM - y3P - 2 * yP + yPx2M - 4 * yPxM) * z \\
&\quad -3 * z * z - 2 * (x2M - 4 * xM)(2 * y3P - 2 * yP + 5 * z) \\
\Delta y3P &= -x2Comb \\
\Delta x3M &= -y2Comb \\
\Delta y2P &= 4 * x2Comb + 4 * xComb1 * yMmP \\
&\quad -3 * xyComb1 - xMmP * Inner1 \\
\Delta xM &= (4/3) * (10 * xyComb1 + (3 * (3 * y2P + y2Px2M \\
&\quad -4 * (y2PxM + 3 * yP + yPx2M - 4 * yPxM))) * yMmP \\
&\quad +(-40 * y2P + 12 * y3P + 28 * yP) * xMmP) - 5 * y2Comb \\
\Delta yP &= -40 * x2 + 2 * (-5 * xMmP2 - 8 * xComb1 * yMmP \\
&\quad +6 * xyComb1 + 2 * xMmP * Inner1) \\
dz &= 2 * (x2Comb + y2Comb) + 7 * xyComb1 \\
&\quad +(31 * x2M - 12 * x3M + 8 * xM - 3 * (y2Px2M \\
&\quad -4 * (y2PxM + yPx2M - 4 * yPxM))) * yMmP \\
&\quad +(31 * y2P - 3 * y2Px2M + 12 * y2PxM - 12 * y3P \\
&\quad +8 * yP + 12 * yPx2M - 48 * yPxM) * xMmP \\
&\quad -9 * (xMmP2 + yMmP2) \\
B0 &= -y2Comb * xComb1 - x2Comb * yComb1 \\
&\quad +xyComb1 * Inner1
\end{aligned}
\tag{B.48}
$$

B.16 Linearization G

$$
\begin{aligned}
P1 &= (1+Z_x^2)*Z_{yy} \\
P2 &= 2*Z_x*Z_y*Z_{xy} \\
P3 &= (1+Z_y^2)*Z_{xx} \\
Z_x &= (X2M-4XM+3Z)/(2\Delta x) \\
Z_{xx} &= (-X3M+4X2M-5XM+2Z)/(\Delta x^2) \\
Z_y &= (Y2M-4YM+3Z)/(2\Delta y) \\
Z_{yy} &= (-Y3M+4Y2M-5YM+2Z)/(\Delta y^2) \\
Z_{xy} &= (((Y2MX2M-4Y2MXM+3Y2M)/(2\Delta x)) \\
&\quad -4((YMX2M-4YMXM+3YM)/(2\Delta x)) \\
&\quad +3((X2M-4XM+3Z)/(2\Delta x)))/(2\Delta y) \\
Z &= Res*8\Delta x^2\Delta y^2
\end{aligned}
\tag{B.49}
$$

$$
\begin{aligned}
xMmP &= x2M - 4 * xM + 3 * z \\
yMmP &= y2M - 4 * yM + 3 * z \\
xComb1 &= 4 * x2M - x3M - 5 * xM + 2 * z \\
yComb1 &= 4 * y2M - y3M - 5 * yM + 2 * z \\
xyComb1 &= xMmP * yMmP \\
Inner1 &= 3 * x2M - 12 * xM + 3 * y2M + y2Mx2M \\
&\quad -4 * (y2MxM + 3 * yM + yMx2M - 4 * yMxM) + 9 * z \\
yComb2 &= 5 * y2M - 2 * y3M + 2 * yM - 5 * z \\
x2Comb &= 8 * \Delta x^2 + 2 * xMmP * xMmP \\
y2Comb &= 8 * \Delta y^2 + 2 * yMmP * yMmP \\
Inner3 &= y2Mx2M - 4 * (y2MxM + 3 * yM + yMx2M - 4 * yMxM) \\
Inner4 &= 3 * (7 * y2M - Inner3 + 4 * (yM - y3M)) \\
Inner5 &= Inner4 * z - 30 * z * z \\
&\quad -(y2M - 4 * yM) * (3 * y2M + Inner3) \\
bot &= 1/(8 * \Delta x^2 * \Delta y^2)
\end{aligned}
\tag{B.50}
$$

$$
\begin{aligned}
\Delta y2Mx2M &= -xyComb1 \\
\Delta y2MxM &= 4 * xyComb1 \\
\Delta yMx2M &= 4 * xyComb1 \\
\Delta yMxM &= -16 * xyComb1 \\
\Delta x2M &= 4 * y2Comb + 2 * (x2M - 4 * xM) * yComb2 + Inner5 \\
\Delta y3M &= -x2Comb \\
\Delta x3M &= -x2Comb \\
\Delta y2M &= 4 * x2Comb + 4 * xComb1 * yMmP \\
&\quad -3 * xyComb1 - xMmP * Inner1 \\
\Delta xM &= -5 * y2Comb - 4 * (-8 * xM * yComb2 \\
&\quad +2 * x2M * (5 * y2M - 2 * y3M + 2 * yM - 5 * z) \\
&\quad +Inner5) \\
\Delta yM &= -5 * x2Comb + 2 * (-8 * xComb1 * yMmP \\
&\quad +6 * xyComb1 + 2 * xMmP * Inner1) \\
dz &= 2 * (x2Comb + y2Comb) - 11 * xyComb1 \\
&\quad +(40 * x2M - 12 * x3M - 28 * xM \\
&\quad -3 * (3 * y2M + Inner3)) * yMmP \\
&\quad +(-9 * x2M + 36 * xM + 40 * y2M - 3 * Inner3 \\
&\quad +4 * (-16 * yM - 3 * y3M)) * xMmP \\
B0 &= y2Comb * xComb1 + x2Comb * yComb1 \\
&\quad -xyComb1 * Inner1
\end{aligned}
\tag{B.51}
$$

Appendix C

High Resolution Images

This appendix contains high resolution images for select figures.

C.1 CFD Images for 2D Validation

This section contains high resolution copies of the images in Figure 3.5. Subplots a-d are presented in Figures C.1 through C.4, respectively.

C.2 Surface Motion Images of 3D Wing

This section contains high resolution copies of the images presented in Figure 4.1. The plots are grouped by time step; wherein the time steps from 0.000 through 0.875 are presented in Figures C.5 through C.12, respectively.

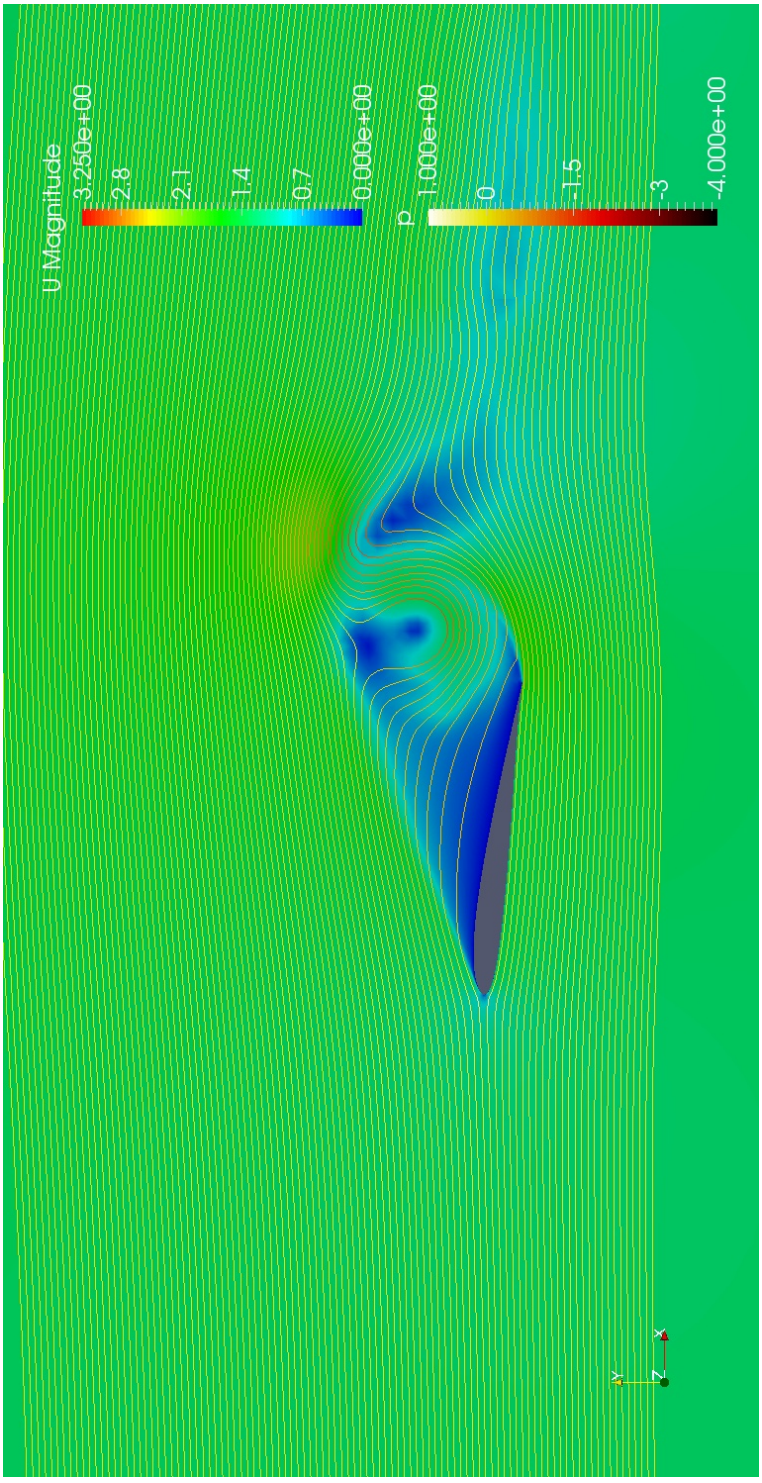


Figure C.1: High Resolution, 2D Validation, t/T = 0.00

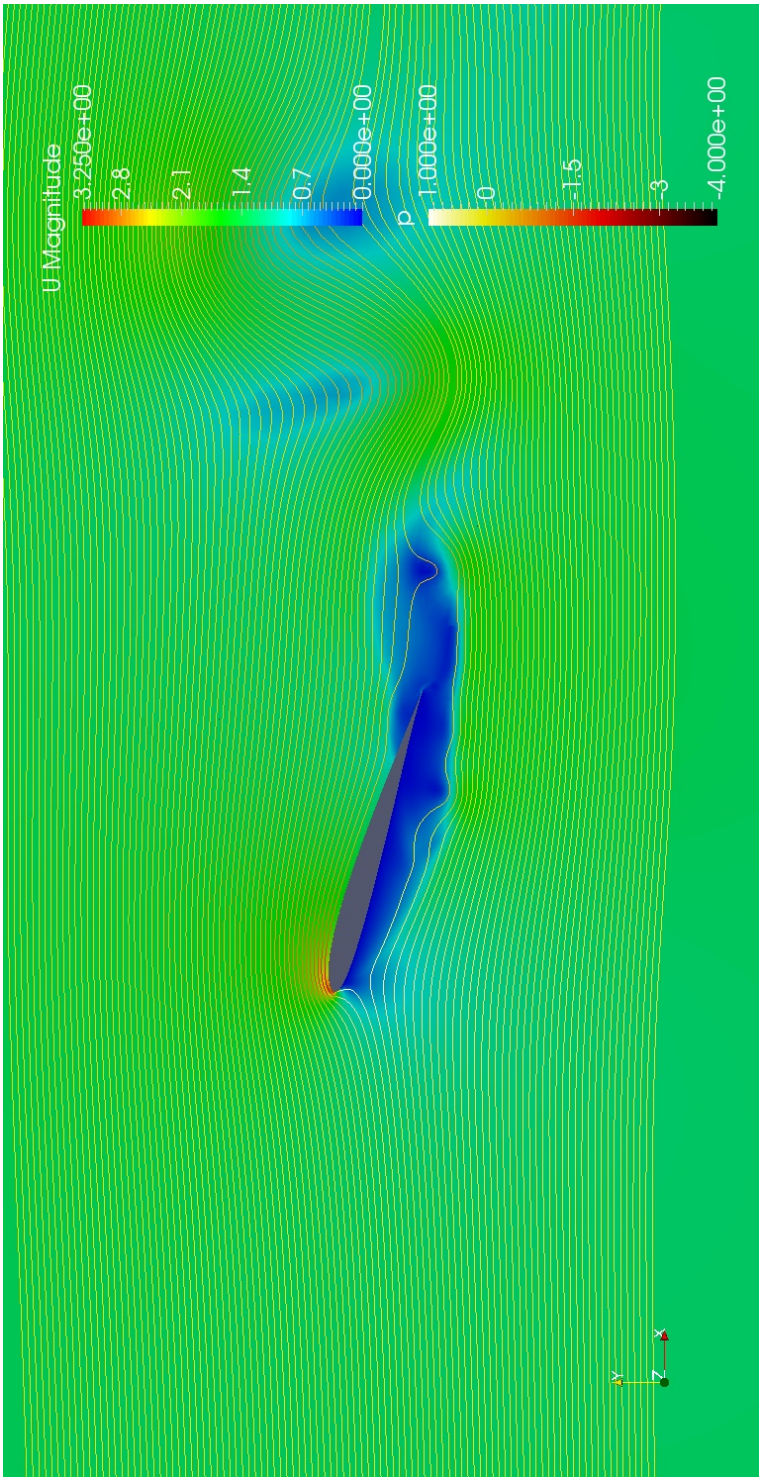


Figure C.2: High Resolution, 2D Validation, t/T = 0.25

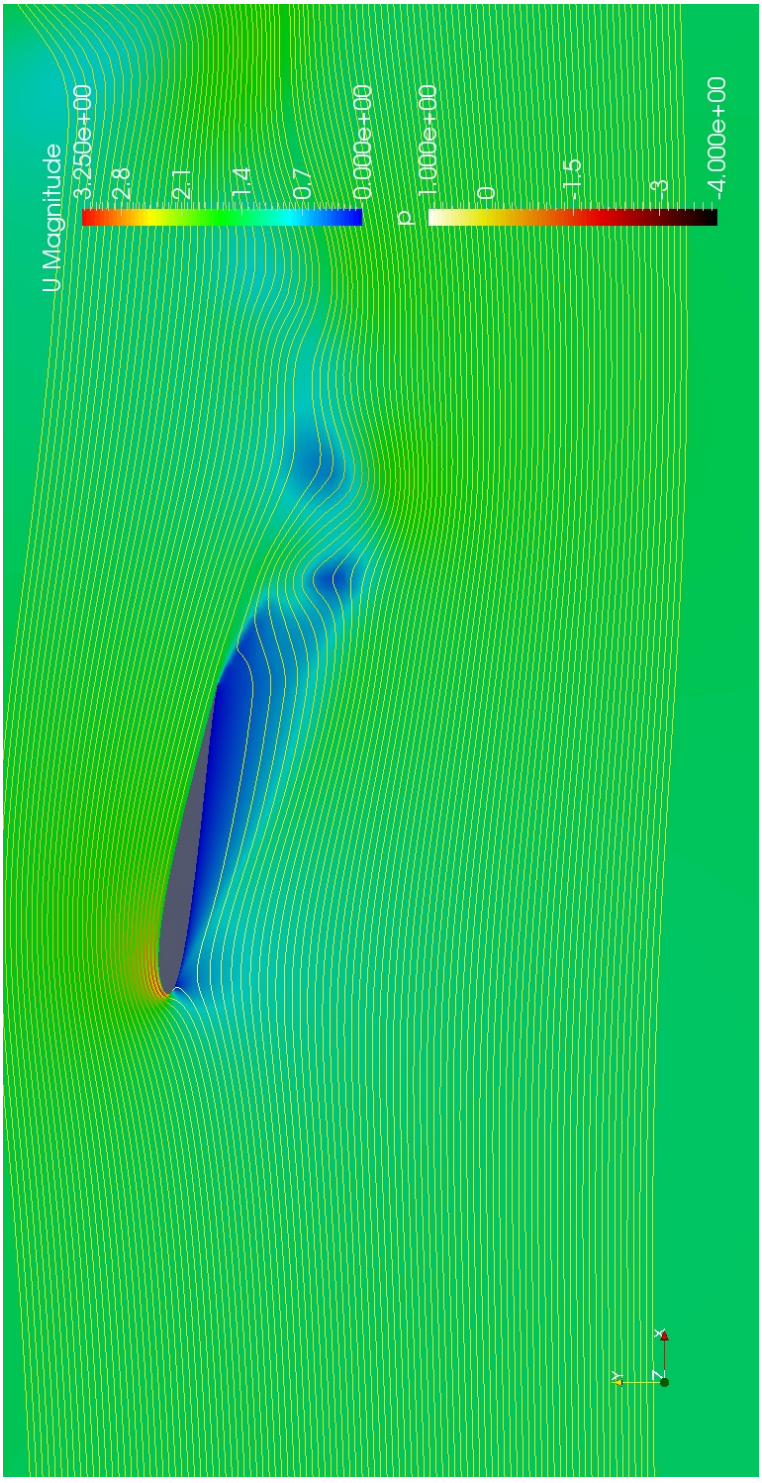


Figure C.3: High Resolution, 2D Validation, t/T = 0.50

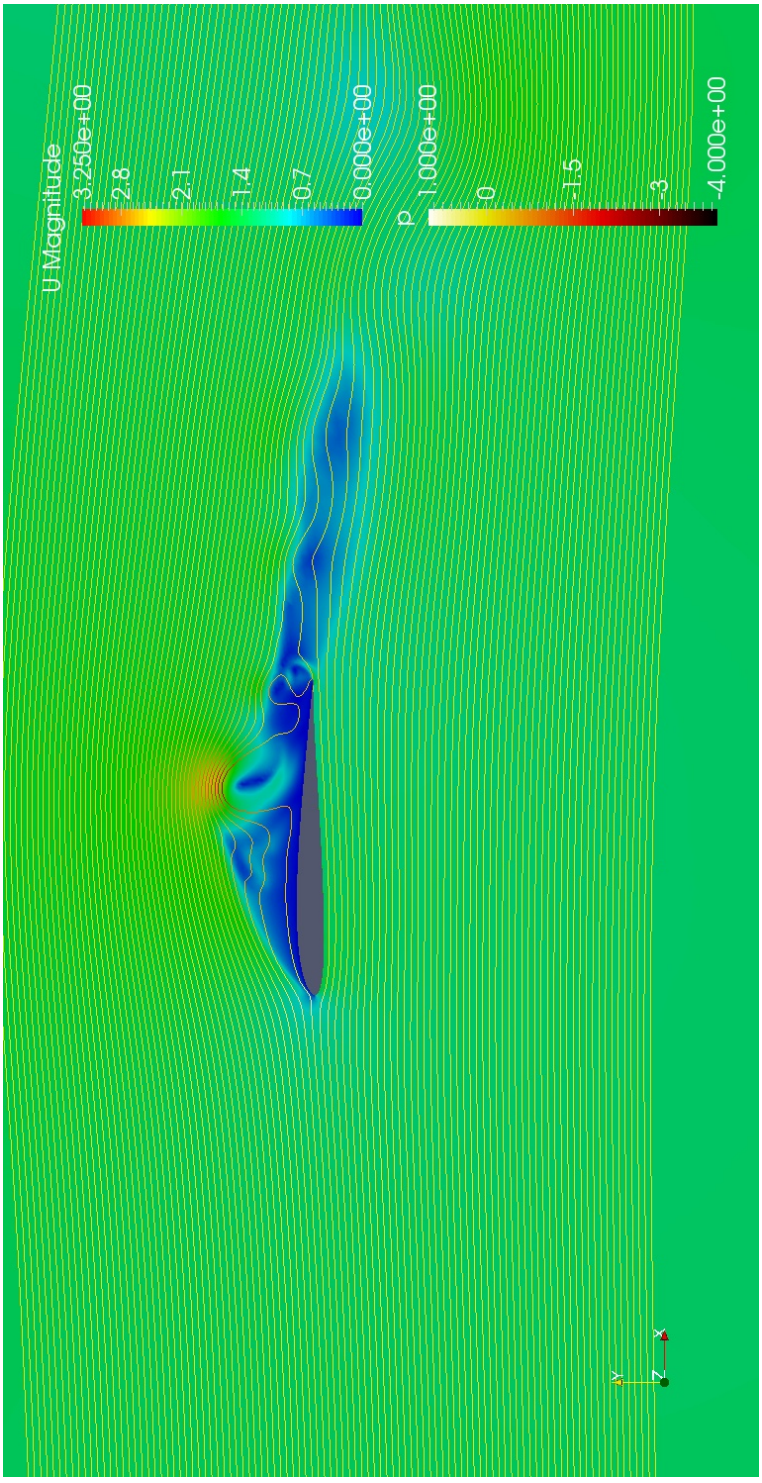
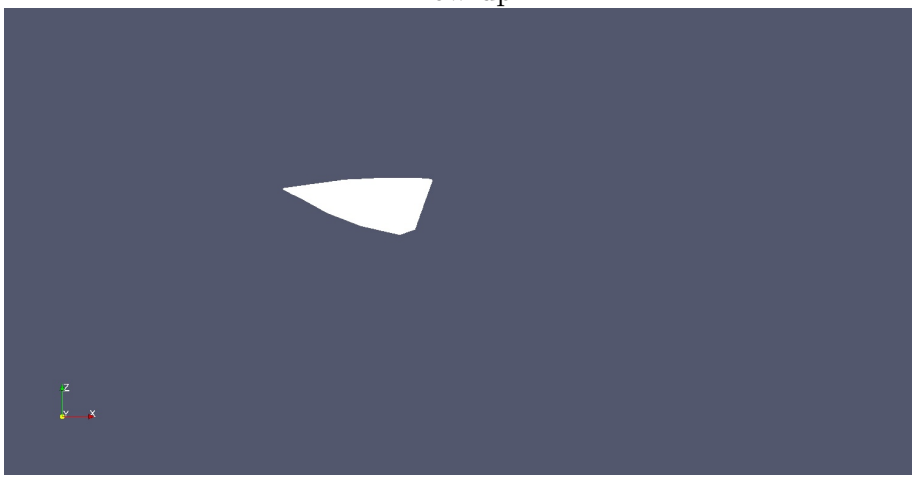


Figure C.4: High Resolution, 2D Validation, t/T = 0.75

Flow up



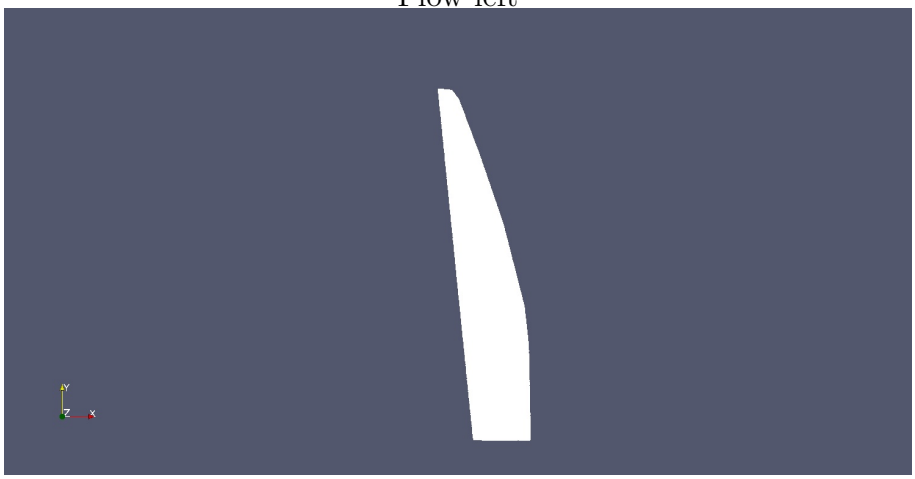
Flow left

Flow left

Figure C.5: High Resolution, 3D Motion, t/T = 0.000

Flow up

Flow left

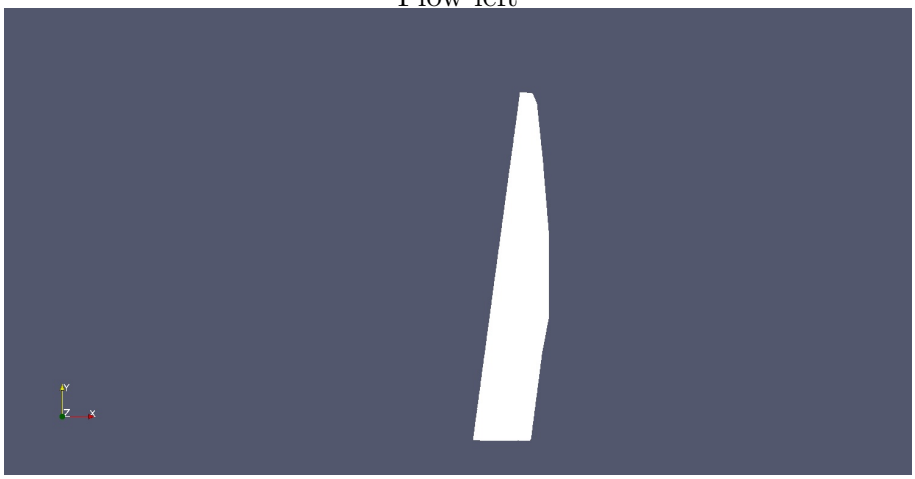


Flow left

Figure C.6: High Resolution, 3D Motion, t/T = 0.125

Flow up

Flow left



Flow left

Figure C.7: High Resolution, 3D Motion, t/T = 0.250

Flow up

Flow left

Flow left

Figure C.8: High Resolution, 3D Motion, t/T = 0.375

Flow up

Flow left

Flow left

Figure C.9: High Resolution, 3D Motion, t/T = 0.500

Flow up

Flow left

Flow left

Figure C.10: High Resolution, 3D Motion, t/T = 0.625

Flow up

Flow left

Flow left

Figure C.11: High Resolution, 3D Motion, t/T = 0.750

Flow up

Flow left

Flow left

Figure C.12: High Resolution, 3D Motion, t/T = 0.875

Vita

Collin Strassburger was born in 1991. During grade school he discovered his passion for flight and robotics. As schooling progressed a greater emphasis was placed on an aerospace education than a robotics education and thus he graduated with his bachelors degree in Aerospace Engineering from Georgia Tech in 2014. While obtaining his bachelors degree he began to observe how many unconventional concepts are available and could prove beneficial if implemented in current designs. To further understand these concepts, he returned to obtain his masters degree in the same field with a focus on applied mechanics from the University of Tennessee. After graduation he hopes to make a difference in the field of aerospace engineering by applying these unconventional concepts to find solutions for problems both conventional and unorthodox.