COMPUTATION OF INTERACTIONAL AERODYNAMICS FOR NOISE
PREDICTION OF HEAVY LIFT ROTORCRAFT

A Dissertation in
Aerospace Engineering
by
Christopher C. Hennes

© 2013 Christopher C. Hennes

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2013
The dissertation of Christopher C. Hennes was reviewed and approved* by the following:

Kenneth S. Brentner  
Professor of Aerospace Engineering  
Dissertation Advisor, Chair of Committee

Sven Schmitz  
Assistant Professor of Aerospace Engineering

Philip J. Morris  
Boeing/A. D. Welliver Professor of Aerospace Engineering

Anthony Atchley  
Professor of Acoustics

George Lesieutre  
Professor of Aerospace Engineering  
Head of the Department of Aerospace Engineering

*Signatures are on file in the Graduate School.
Abstract

Many computational tools are used when developing a modern helicopter. As the design space is narrowed, more accurate and time-intensive tools are brought to bear. These tools are used to determine the effect of a design decision on the performance, handling, stability and efficiency of the aircraft. One notable parameter left out of this process is acoustics. This is due in part to the difficulty in making useful acoustics calculations that reveal the differences between various design configurations. This thesis presents a new approach designed to bridge the gap in prediction capability between fast but low-fidelity Lagrangian particle methods, and slow but high-fidelity Eulerian computational fluid dynamics simulations. A multi-pronged approach is presented. First, a simple flow solver using well-understood and tested flow solution methodologies is developed specifically to handle bodies in arbitrary motion. To this basic flow solver two new technologies are added. The first is an Immersed Boundary technique designed to be tolerant of geometric degeneracies and low-resolution grids. This new technique allows easy inclusion of complex fuselage geometries at minimal computational cost, improving the ability of a solver to capture the complex interactional aerodynamic effects expected in modern rotorcraft design. The second new technique is an extension of a concept from flow visualization where the motion of tip vortices are tracked through the solution using massless particles convecting with the local flow. In this extension of that concept, the particles maintain knowledge of the expected and actual vortex strength. As a post-processing step, when the acoustic calculations are made, these particles are used to augment the loading noise calculation and reproduce the highly-impulsive character of blade-vortex interaction noise. In combination these new techniques yield a significant improvement to the state of the art in rotorcraft blade-vortex interaction noise prediction.
# Table of Contents

List of Figures .................................................. viii
List of Symbols ................................................. xiii
Acknowledgments .................................................. xvi

Chapter 1  Introduction ........................................ 1
  1.1 Overview of Deficiencies in Current Techniques .......... 2
  1.2 Review of Noise Prediction Technique .................... 7
  1.3 Rotorcraft Noise Sources ................................ 12
  1.4 Summary of Innovations .................................. 16

Chapter 2  Literature Review .................................. 17
  2.1 Immersed Boundary CFD Methods ......................... 17
  2.2 Literature Review of Vorticity Preservation Techniques 24
    2.2.1 Lagrangian Solvers ................................ 25
    2.2.2 Navier-Stokes Eulerian Solvers .................... 27
      2.2.2.1 Vorticity Confinement ....................... 28
      2.2.2.2 Vorticity Transport Solvers ................. 29
    2.2.3 Coupled Lagrangian/Eulerian Techniques .......... 29

Chapter 3  Immersed Boundary Fuselage Representation ...... 31
  3.1 Flow Solver Overview ................................... 32
  3.2 Governing Equations .................................... 33
    3.2.1 Explicit Method .................................. 34
    3.2.2 Implicit Method .................................. 36
    3.2.3 Artificial Viscosity .............................. 38
    3.2.4 Comparison to OVERFLOW ....................... 40
3.3 Oversetting ................................................. 42
3.4 Automatic Off-Body Grid Generation .................... 43
3.5 Load Balancing ............................................ 44
3.6 Support for CAD Input .................................... 46
  3.6.1 Common Errors in Discretized Geometry .......... 48
  3.6.2 Efficient Triangulated Surface Storage and Access .. 51
  3.6.3 Interior-Exterior Testing ............................ 53
  3.6.4 Robust Intersection Testing ....................... 54
  3.6.5 Surface Curvature Calculation .................... 58
  3.6.6 Closest Surface Point Identification ............... 59
3.7 Treatment of Immersed Boundaries ...................... 61
  3.7.1 General Advantages and Disadvantages of Penalization Methods .............................................. 63
  3.7.2 Penalization Method 1: Discrete Forcing .......... 68
  3.7.3 Penalization Method 2: Volume Fraction .......... 75
  3.7.4 Penalization Method 3: Velocity Mirroring ........ 80
  3.7.5 General Advantages and Disadvantages of Reconstruction Methods ........................................... 86
  3.7.6 Reconstruction Method 1: Surface Normal Extrapolation ......................................................... 86
    3.7.6.1 Correction for Unresolved Body Features ...... 90
    3.7.6.2 No-Slip Wall Results ........................... 93
    3.7.6.3 Slip Wall Results ................................ 98
  3.7.7 Reconstruction Method 2: Inverse Distance Weighted Average .................................................... 103
    3.7.7.1 No-Slip Wall Results ........................... 106
    3.7.7.2 Slip Wall Results ................................ 110
    3.7.7.3 Coarse Grid Analysis ........................... 115
    3.7.7.4 Stationary 2D NACA 0009 angle of attack sweep 118
    3.7.7.5 Three-Dimensional Analysis .................... 123
  3.7.8 Computational Cost .................................. 125

Chapter 4 Vortex Interaction Strength Preservation ...... 128
  4.1 Necessity of Vortex Interaction Strength Preservation ................................................................. 129
    4.1.1 Modern High-Fidelity CFD .......................... 130
  4.2 Mathematical Model of Vortex and Convection ........ 134
  4.3 Description of New Method ............................. 135
    4.3.1 Implementation Details ............................. 140
    4.3.2 Vortex-Induced Velocity Field .................. 143
  4.4 Calculation of Vortex Strength ......................... 143
    4.4.1 Direct calculation of circulation ................ 144
4.4.2 Direct calculation of circulation and roll-up ........................................ 146
4.4.3 Approximation of circulation using bound circulation ...................... 146
4.5 Vortex Formation, Growth, and Destruction ........................................ 147
4.6 Particle Tracking Demonstration Cases .............................................. 150
  4.6.1 Ring Vortex .................................................................................. 150
  4.6.2 2D Vortex-Airfoil Interaction .......................................................... 153
  4.6.3 3D Vortex-Rotor Interaction ............................................................. 155
  4.6.4 HART-II ....................................................................................... 165
  4.6.5 Tiltrotor ........................................................................................ 177

Chapter 5 Conclusions and Future Work ................................................... 189
  5.1 Surface Treatment Techniques ............................................................ 191
  5.2 Geometry Handling ........................................................................... 192
  5.3 Blade-Vortex Interaction Capturing .................................................... 193
  5.4 Code Speed Optimization .................................................................. 194
  5.5 Conclusions ....................................................................................... 195

Appendix IBSEN User’s Manual ................................................................. 197
  A.1 Compiling and Installing IBSEN ......................................................... 197
  A.2 File Overview and Basic Setup .......................................................... 201
  A.3 CaseDefinition.xsd .......................................................................... 201
  A.4 Main input file .................................................................................. 202
    A.4.1 Header ....................................................................................... 202
    A.4.2 <case> Element Inputs ............................................................... 204
    A.4.3 Intermittent Controller Inputs ...................................................... 211
    A.4.4 Immersed Objects ...................................................................... 213
      A.4.4.1 <sphere> ............................................................................ 215
      A.4.4.2 <box> .................................................................................. 216
      A.4.4.3 <cylinder> .......................................................................... 216
      A.4.4.4 <stlobject> ......................................................................... 217
      A.4.4.5 <plot3dobject> ................................................................... 219
    A.4.5 Motions ....................................................................................... 220
      A.4.5.1 <aperiodictranslation> ............................................................ 221
      A.4.5.2 <periodictranslation> ............................................................... 222
      A.4.5.3 <periodicrotation> ................................................................. 223
      A.4.5.4 <periodicrotationfromfile> .................................................... 224
      A.4.5.5 <aperiodicrotation> .............................................................. 225
      A.4.5.6 <copymotionfrom> ................................................................. 226
    A.4.6 Elastic Motions ............................................................................ 226
      A.4.6.1 <discretebendingdeformation> ............................................. 227
List of Figures

1.1 Advanced rotorcraft configurations. .................................................. 15

3.1 Outline of important equivalent OVERFLOW parameters when running IBSEN. ................................................................. 42

3.2 An example auto-generated grid. .......................................................... 44

3.3 Line segment AB is directly checked for intersection with the geometry surface, rather than relying on the interior/exterior calculation to succeed at point B. .......................................................... 50

3.4 An example of inside/outside detection using ray-casting. The upper ray intersects the geometry an odd number of times (once) indicating that the originating point is interior, while the lower ray intersects the geometry an even number of times (twice), indicating that the originating point is exterior. .......................................................... 55

3.5 Degenerate cases in penalization methods. .............................................. 65

3.6 Innermost grid configuration for boundary condition test cases. .......... 68

3.7 Stencil with multiple points inside the body allowing the velocity of the interior of the solid body to influence the flowfield, resulting in a non-zero velocity at the actual body/fluid interface. ............... 70

3.8 Lift coefficient vs. time for discrete-forcing penalization cases. Dashed line shows nominal thin-arifoil result, $C_l = 0.33$ ............. 72

3.9 Pressure coefficient distribution for the discrete-forcing penalization case. Solid line is the inviscid OVERFLOW solution, symbols are the IBSEN solution. .......................................................... 73

3.10 Pressure field for discrete-forcing penalization case. ......................... 74

3.11 The volume fraction technique modifies the force applied to more accurately reflect the actual physical location of the body surface. .... 76

3.12 Lift coefficient vs. time for volume fraction penalization cases. Dashed line shows nominal thin-arifoil result, $C_l = 0.33$ ............. 77

3.13 Pressure coefficient distribution for the volume fraction penalization case. Solid line is the inviscid OVERFLOW solution, symbols are the IBSEN solution. .......................................................... 78
3.14 Pressure field for volume fraction penalization case. 79
3.15 Schematic of the velocity mirroring technique showing the velocity at the body surface forced to zero in the differentiation by reflecting the velocity across the surface. 81
3.16 Lift coefficient vs. time for mirroring penalization cases. Dashed line shows nominal thin-arifoil result, $C_l = 0.33$. 83
3.17 Pressure coefficient distribution for the mirroring penalization case. Solid line is the inviscid OVERFLOW solution, symbols are the IBSEN solution. 84
3.18 Pressure field for mirroring penalization case. 85
3.19 Surface normal extrapolation reconstruction technique. 87
3.20 The surface normal extrapolation reconstruction technique will fail in under-resolved regions of high concavity due to the inability to interpolate to any of points C–F. 91
3.21 With the non-resolved points filled in, the body appears to the flow solution procedure to be smooth over the top of the unresolved cavity. 92
3.22 Lift coefficient vs. time for the no-slip wall SNE reconstruction cases. Dashed line shows nominal thin-arifoil result, $C_l = 0.33$. 95
3.23 Pressure coefficient distribution for the no-slip wall SNE reconstruction case. Solid line is the inviscid OVERFLOW solution, symbols are the IBSEN solution. 96
3.24 Pressure field for the no-slip wall SNE reconstruction case. 97
3.25 Lift coefficient vs. time for the slip wall SNE reconstruction cases. Dashed line shows nominal thin-arifoil result, $C_l = 0.33$. 100
3.26 Pressure coefficient distribution for the slip wall SNE reconstruction case. Solid line is the inviscid OVERFLOW solution, symbols are the IBSEN solution. 101
3.27 Pressure field for the slip wall SNE reconstruction case. 102
3.28 Schematic of the Inverse Distance Weighted Average method. 104
3.29 Lift coefficient vs. time for the no-slip wall IDWA reconstruction cases. Dashed line shows nominal thin-arifoil result, $C_l = 0.33$. 107
3.30 Pressure coefficient distribution for the no-slip wall IDWA reconstruction case. Solid line is the inviscid OVERFLOW solution, symbols are the IBSEN solution. 108
3.31 Pressure field for the no-slip wall IDWA reconstruction case. 109
3.32 Lift coefficient vs. time for the slip wall IDWA reconstruction cases. Dashed line shows nominal thin-arifoil result, $C_l = 0.33$. 112
3.33 Pressure coefficient distribution for the slip wall IDWA reconstruction case. Solid line is the inviscid OVERFLOW solution, symbols are the IBSEN solution. ........................................ 113
3.34 Pressure field for the slip wall IDWA reconstruction case. ........ 114
3.35 Comparison of pressure fields for various grid densities using the IDWA immersed boundary technique. .................... 117
3.36 Example of the overset grid system used in the NACA 0009 angle of attack sweep. .................................................. 118
3.37 Inner grid showing level of surface resolution. ...................... 119
3.38 Contours of pressure coefficient calculated by IBSEN for a NACA 0009 airfoil at 3 degrees angle of attack. .................... 120
3.39 Contours of pressure coefficient calculated by OVERFLOW for a NACA 0009 airfoil at 3 degrees angle of attack (inviscid result). 120
3.40 Momentum vectors and velocity magnitude contours at the leading edge of a NACA 0009 at 3 degrees angle of attack. IBSEN prediction. ......................................................... 121
3.41 Comparison of IBSEN and OVERFLOW surface pressure predictions for a NACA 0009 airfoil at 3 degrees angle of attack. ...... 122
3.42 Coefficient of lift versus angle of attack for IBSEN NACA 0009 run, compared to experimental results at Re=360,000. ........... 123
3.43 ROBIN fuselage geometry. ........................................... 124
3.44 Immersed Boundary near-body grids used in ROBIN analysis. ... 126
3.45 Pressure contours on a plane through the center of the ROBIN fuselage, $M_\infty = 0.2$. ..................................................... 127

4.1 HELIOS vortex position prediction for four different grids (Figure 11 from Ref. 10). Red spheres show experimental position measurements. ......................................................... 132
4.2 Comparison of experimental and predicted vortex strengths (Figures 14 and 15 from Ref. 10). ........................................... 133
4.3 Vortex velocity profiles for a potential and a Scully-model vortex, $\Gamma = 1.0$, $r_c = 0.1$. ......................................................... 135
4.4 Example of vortex particle position integration. ..................... 140
4.5 Example of representation of a line vortex. ......................... 141
4.6 Selection of vortex filaments to use in surface pressure calculation. 141
4.7 Core radius versus wake age (Figure 10.22 from Ref. 134). ....... 149
4.8 Vortex ring convection in free-vortex solver. Solid lines are the vortex filament location; symbols are the massless particle positions. ......................................................... 152
4.9 Vortex ring convection in IBSEN. ...................................... 154
4.10 Schematic of 2D BVI case setup (in actual case vortex was placed 30 chords upstream). .............................................. 155
4.11 Vortex circulation strength evaluated on a circle $3R_c$ from the vortex center. .................................................. 156
4.12 Side view of vortex interaction case setup showing the upstream vortex generator and small downstream rotor. ............... 157
4.13 Side view of a slice through the middle of the 3D Vortex-Rotor Interaction overset grid system. .............................. 158
4.14 Isosurfaces of vorticity colored by local pressure. ............. 161
4.15 Two views of the massless particle convection, colored by local circulation (calculated on a circle of radius $10r_c$). ............... 162
4.16 Eulerian prediction of the local and bound circulation at the particle locations at a single instant in time, as a function of the particles’ distance from the rotor center. .............................. 163
4.17 Slices of vorticity magnitude at three different downstream positions during the formation of the tip vortex. ..................... 164
4.18 HART case setup used in the with-fuselage simulations. ........ 166
4.19 Top view of the HART-II overset grid system. ....................... 167
4.20 HART II body-fitted blade mesh slice. .............................. 167
4.21 Comparison of tip vortex positions between the isolated (red circles) and interaction (blue squares) cases. ........................ 169
4.22 Magnitude of the local acoustic loading vector over one rotor revolution for a representative blade in the HART-II simulation, with and without fuselage, and with and without vortex particle reconstruction of the BVI events. $\Psi = 0^\circ$ is at the top of the plots. 172
4.23 BVISPL in dB re:20$\mu$Pa (calculated for harmonics 4–40) for the baseline DP128 case, experimental and theoretical results, from Ref. 137. $\Psi = 0^\circ$ is at the top of the plots. ........................... 173
4.24 BVISPL in dB re:20$\mu$Pa (calculated for harmonics 4–40) for the baseline DP128 case, experimental and IBSEN results at the same scale, baseline CFD case with no BVI reconstruction. $\Psi = 0^\circ$ is at the top of the plots. .............................. 174
4.25 BVISPL in dB re:20$\mu$Pa (calculated for harmonics 4–40) for the baseline DP128 case, rescaled to show detail. $\Psi = 0^\circ$ is at the top of the plots. .................................................. 175
4.26 BVISPL in dB re:20$\mu$Pa (calculated for harmonics 4–40) for the baseline DP128 case, experimental and IBSEN results at the same scale, CFD case with added particle-based BVI reconstruction. $\Psi = 0^\circ$ is at the top of the plots. .............................. 176
4.27 Notional tiltrotor blade geometry used in the tiltrotor analysis... 178

xi
4.28 Tiltrotor fuselage model from http://www.3dcadbrowser.com. . 179
4.29 Tiltrotor fuselage model detail. ................................. 180
4.30 Rotor centerline slice showing the grid system used in the tiltrotor analysis. ......................................................... 181
4.31 Snapshot of momentum magnitude on a slice through the rotor disks after seven rotor revolutions. .................. 183
4.32 Snapshot of tip vortex positions after seven rotor revolutions. . . . . . 184
4.33 Snapshot of tip vortex strengths after seven rotor revolutions. . . . . . . 185
4.34 Overlay of tip vortex positions after seven rotor revolutions. Red line is interaction case, blue is isolated. .................. 186
4.35 Snapshot of tip vortex positions after seven rotor revolutions, with burst filaments removed. .................. 187
4.36 BVISPL comparison for the tiltrotor case. .................. 188

A.1 Example input script for an MPI run on a cluster. Note that clusters vary and it is likely your script will look somewhat different. 258
A.2 Using the “tail” command to follow along with the output. The “-f” will continuously refresh the output as the file is written to. . 259
## List of Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Box^2$</td>
<td>d’Alembertian or wave operator, $\Box^2 = \left[ (1/c^2) \left( \partial^2 / \partial t^2 \right) \right] - \nabla^2$</td>
</tr>
<tr>
<td>$c$</td>
<td>Speed of sound, ms$^{-1}$</td>
</tr>
<tr>
<td>$C$</td>
<td>Confinement strength</td>
</tr>
<tr>
<td>$dS$</td>
<td>Differential element of surface area, m$^2$</td>
</tr>
<tr>
<td>$f = 0$</td>
<td>Function that describes the integration surface (i.e. the rotor blade)</td>
</tr>
<tr>
<td>$f_i$</td>
<td>Forcing function used to model body in immersed boundary method, N</td>
</tr>
<tr>
<td>$g$</td>
<td>Retarded time function, $g = \tau - t + r/c$, s</td>
</tr>
<tr>
<td>$h$</td>
<td>Grid spacing parameter, m</td>
</tr>
<tr>
<td>$H$</td>
<td>Heaviside function, $H(f) = 0$ for $f &lt; 0$ and $H(f) = 1$ for $f &gt; 0$</td>
</tr>
<tr>
<td>$I$</td>
<td>Identity matrix.</td>
</tr>
<tr>
<td>$k_i$</td>
<td>Direction and strength vector for confinement term, m/s</td>
</tr>
<tr>
<td>$\ell_i$</td>
<td>Components of loading force acting on the fluid, Nm$^{-2}$</td>
</tr>
<tr>
<td>$L_i$</td>
<td>The redefined “loading” vector, equal to $\ell_i$ for impermeable surfaces. $L_i = P_{ij}\hat{n}_j + \rho u_i(u_n - v_n)$, Nm$^{-2}$.</td>
</tr>
<tr>
<td>$M$</td>
<td>Mach number of source, $v/c$</td>
</tr>
<tr>
<td>$\hat{n}$</td>
<td>Outward-facing unit normal vector to surface $f = 0$, unit vector in direction of maximum vorticity</td>
</tr>
</tbody>
</table>
$P_{ij}$ Compressive stress tensor, $P_{ij} = T_{ij} - p_0 \delta_{ij}$

$p$ Flowfield pressure, Pa

$p'$ Acoustic pressure, $p - p_0$, Pa

$p'_T$ Thickness noise contribution to $p'$, Pa

$p'_L$ Loading noise contribution to $p'$, Pa

$Q$ Flow solution vector $(\rho, \rho u, \rho v, \rho w)$

$r \ |x_i - y_i|$, Distance between observer and source, m

$r_c$ Vortex viscous core radius, m

$t$ Observer time, s

$T_{ij}$ Lighthill stress tensor, $T_{ij} = \rho u_i u_j + (p - \rho c_0^2) \delta_{ij}$

$u_i$ Velocity of the fluid, $(u, v, w)$, ms\(^{-1}\)

$U_i$ Redefined “velocity” vector, $U_i = \frac{\rho}{\rho_0} (u_i - v_i) + v_i$, ms\(^{-1}\)

$(U, V, W)$ Contravariant velocities, ms\(^{-1}\)

$V_\theta$ Vortex rotational velocity, ms\(^{-1}\)

$V_r$ Vortex radial velocity, ms\(^{-1}\)

$v_i$ Velocity of the blade surface, ms\(^{-1}\)

$x_i$ Observer location, m

$y_i$ Source location, m

**Greek symbols**

$\delta$ Dirac delta function

$\epsilon$ Confinement coefficient

$\Gamma$ Vorticity, s\(^{-1}\)
\( \mu \) Diffusion coefficient

\( \omega \) Vorticity, \( \nabla \times \vec{u}, 1/s \)

\( \sigma_{ij} \) Stress tensor

\( \rho \) Density, kg m\(^{-3}\)

\( \tau \) Source time, s

\( \Psi \) Rotor azimuth

\( \zeta, \eta, \zeta \) Computational coordinate system

**Subscripts**

0 Quantity in the quiescent medium

\( i \) \( i^{th} \) component of a vector

\( L \) Loading noise component

\( n \) Vector quantity taken in the surface normal direction, \( \vec{x} \cdot \hat{n} \)

\( r \) Vector quantity taken in the radiation direction, \( \vec{x} \cdot \hat{r} \)

ret Quantity evaluated at retarded time \( \tau \)

\( T \) Thickness noise component

**Note** – The summation convention is used in this document wherever repeated subscripts appear.
Acknowledgments

It (almost) goes without saying that none of this research would have been possible without the support, both financial and moral, of my advisor Dr. Kenneth Brentner. His patience and forbearance was invaluable as I took far longer than originally anticipated to complete this work. Thanks are also due to my committee for their assistance and support.

In a process as long as this, it is natural that many people helped me along the way: fellow students, other faculty, and of course Penn State’s incomparable support staff. Thanks are especially due to T. Reed for her continued advice throughout. Thanks also to Earl Duque and Chris Stone for the ROBIN grids and OVERFLOW setup assistance, and to Doug Boyd for the HART-II fuselage model.

Finally, without my wife Karen’s continued encouragement I’m sure I would never have had the fortitude to complete this process.

This research was funded in part by the U.S. Government under Agreement No. W911W6-06-2-0008. The U.S. Government is authorized to reproduce and distribute reprints notwithstanding any copyright notation thereon. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.
Chapter 1

Introduction

Many computational tools are used when developing a modern helicopter. Major systems of the aircraft are analyzed in the preliminary design phase using fast, approximate techniques. As the design space is narrowed, more accurate and time-intensive tools are brought to bear on the remaining configurations. These tools are used to determine the effect of a design decision on the performance, handling, stability and efficiency of the aircraft. One notable parameter left out of this process is acoustics. It is rare for acoustics to be considered during the design phase of a helicopter, due in part to the difficulty in making useful acoustics calculations that reveal the differences between various design configurations. In modern rotorcraft design, acoustic predictions are only attempted once the configuration is relatively near the end of the design cycle. Unfortunately, at this point it is difficult and expensive to make changes if problems are identified.

Accurate calculation of the acoustics of a helicopter is a very challenging task. It requires prediction of time-accurate blade aerodynamics, including well-resolved blade tip vortices. This is especially true for modern configurations
where, in an attempt to reduce hub drag, designs feature main rotors very close to the fuselage, increasing the importance of interactional aerodynamics on main rotor noise. In addition to the noise generated by the main rotor and its interactional aerodynamic phenomena, tail rotor noise is impossible to predict without considering the flow interactions, including main rotor tip vortices interacting with the tail rotor blades. Furthermore, new rotorcraft concepts—particularly in the heavy-lift design space—often involve coaxial rotors, pusher-props, tiltrotors, ducted fans, etc., all of which are expected to have extremely complex interactional aerodynamics, and by extension, acoustics.

1.1 Overview of Deficiencies in Current Techniques

Sophisticated modern computational fluid dynamics (CFD) methods are capable of calculating many of these interactions, given enough computer power and engineer time to properly set up and run the simulation. However, turnaround time is critical when dozens or hundreds of possible configurations must be considered: the human time to set up a case as well as the computational time to actually calculate and post-process the flow solution must be considered. While our current high-end algorithms are often theoretically capable of conducting at least the basic aerodynamics for these simulations, the budget does not exist for analyzing rotor noise in this level of detail. However, the analysis tools used to compute rotor aerodynamics span a wide spectrum of techniques of varying levels of computational speed and fidelity, and many tools exist that are fast enough to be used in a design environment, at least in the later stages, at a reduced simulation fidelity.
Throughout the life cycle of a helicopter, engineers use Comprehensive Analysis tools such as RCAS (Ref. 1), CAMRAD II (Ref. 2), and DYMORE (Ref. 3) to analyze the performance, loads, vibration, response, and stability of the system. These tools provide a useful data set for those tasks, but their usefulness for acoustics predictions is limited by their aerodynamic models, generally lifting-line methods with 2D airfoil data lookup tables (Ref. 4). Constantly increasing in sophistication, these tools also require substantial expertise and engineer time to set up and run. For acoustics predictions these codes are most useful for either low-fidelity isolated-rotor studies, or as advanced structural models coupled with more sophisticated aerodynamic prediction tools (Ref. 5). In some cases these codes include panel models of the fuselage (see, e.g. Ref 6), but their Lagrangian formulation with a (relatively) small number of evaluation points limits the fidelity of their predictions of interactional aerodynamics.

At the next fidelity level are CFD codes such as Rot3DC (Ref. 7). These tools generally use coarse non-body-conforming grids for the fuselage and viscous, incompressible Eulerian aerodynamics models. Rot3DC, designed for ease of use and quick case setup and run time, uses a time-averaged momentum disk model of the rotor. This greatly reduces that code’s usefulness in providing an input to acoustic prediction codes, which require time-accurate simulation of the individual blade loading. While coarse acoustics predictions have been carried out using the basic Rot3DC (Ref. 8), considerable further research is required to refine the predictions. Work is underway to include a discrete blade model into that code, and with compressibility corrections it is hoped that Rot3DC may provide a basic level of acoustic prediction capability for complex
heavy-lift configurations. While this technique is an order of magnitude slower than Lagrangian free-vortex analyses, it is still considerably faster than the next level of fidelity, discussed below.

At the far end of the spectrum of tools are the full compressible, viscous, body-fitted Navier-Stokes codes such as OVERFLOW (Ref. 9). These codes typically use extremely sophisticated CFD technology and are capable of predicting rotorcraft interactional aerodynamic effects reasonably well, particularly in combination with recently-developed high-order spatial and temporal accuracy schemes when used in combination with high-resolution (potentially dynamically adaptive) grids. One challenge with these tools lies in case setup. Development of a system of structured, body-fitted overset grids for use in an OVERFLOW simulation requires considerable expertise and experience, as well as a substantial engineer time requirement. Recent research has pushed the boundaries of what these codes can compute. By using adaptive mesh refinement (AMR) techniques researchers have been able to compute and convect rotor tip vortices and other complex flow structures in unprecedented detail and improved vortex strength—though still far below experimentally-measured strengths—at the price of hundreds of thousands of hours of computer time and using hundreds of millions or even billions of grid points, even for “simple” isolated rotor cases (Ref. 10). Including the fuselage in these simulations typically results in another substantial increase in the number of grid points required and thus the computational requirements.

In order for acoustics predictions to be a useful helicopter design tool, they must be able to accurately distinguish between the noise generated by the various configurations under consideration. For a modern helicopter this requires
high-fidelity Navier-Stokes simulations in order to capture the influence of the fuselage on the basic rotor inflow as well as the propagation of the rotor tip vortices. It is unrealistic, however, to imagine running multiple billion-point CFD simulations during the design of a new rotorcraft—techniques must be developed to reduce this onerous computational requirement, and yet preserve both the inclusion of the interactional aerodynamics effects from the fuselage on the rotor inflow, as well as the tip vortex position and strength.

Immersed Boundary techniques present a promising technique for the inclusion of the fuselage in OVERFLOW-type simulations when the goal of the simulations is acoustic predictions. These techniques are similar to the technique used to include the fuselage in Rot3DC. They do not require the development of a body-fitted fuselage mesh, and the attendant increase in grid points and case setup time that entails. The price of the technique is that, for the grid densities under consideration in this research, fuselage surface pressure predictions are not expected to be adequate for other uses (e.g. drag prediction, etc.). Since the acoustics predictions are only concerned with the pressure on the blade surfaces, however, this is an acceptable compromise, provided that the influence of the fuselage on the overall rotor inflow and rotor wake development can be adequately captured.

The vast majority of current Immersed Boundary techniques are viscous, incompressible techniques (see Ref. 11 for a review), although some work in compressibility is ongoing (see e.g. Ref. 12), and at least one team of researchers presents an analysis of the method applied to inviscid flow calculations (Ref. 13). The principle disadvantage of immersed boundary flows is that the surface treatment methods currently available generally require very fine
grids for stability and accuracy. All immersed boundary research to date has focused on refining the technique through increasingly sophisticated techniques applied to very fine (viscous-spacing) grids and frequently simple (sometimes 2D-only) geometries, neglecting any possible degeneracies (see Chapter 2 for further information). The research in this dissertation takes the opposite approach, first developing a baseline capability to solve flow on extremely complex geometries in arbitrary motion. A grid system that is representative of the targeted resolution necessary for reasonable solver runtime is created, and then grids of this resolution are used to examine various immersed boundary techniques.

An additional challenge to grid-based Navier-Stokes methods (as opposed to Lagrangian discrete-vortex methods) is the tendency of Eulerian solvers to dissipate the rotor tip vortices much more quickly than occurs in reality. Accurate computation of the position and strength of the blade-tip vortices is crucial for the computation of unsteady loading in certain flight conditions; that unsteady loading is an important, sometimes overwhelmingly dominant, source of interaction noise generation. Various techniques have been developed in an attempt to make these tools more suitable for Blade-Vortex Interaction (BVI) prediction. Researchers are actively developing techniques to couple grid-based solvers in the near-field with vortex particles, filaments, or blobs in the far-field for the long-time convection of the tip vortices (Refs. 14). Other researchers have attempted to solve the fluid equations of motion in a vorticity-conserving manner by using the vorticity transport equations (e.g., Refs. 15–18). Most of these techniques are still in their infancy, continue to require the complicated setup of conventional body-fitted solvers, and have not been evaluated for their noise-
prediction capabilities. Other research has focused on higher-order methods and the use of adaptive mesh refinement to improve the predictions. While results are promising, the computational requirements put these methods far out of reach for acoustics predictions during rotorcraft design with present computational capabilities.

A new approach is needed that enables complex aerodynamics and acoustics to be included earlier in the design phase, when modifications to the design are still relatively inexpensive. What is missing from the currently available tools is a tool that is easy to set up (like Rot3DC), which is capable of capturing the complex time-dependent interactional aerodynamics and flow around the rotor blades and fuselage (like OVERFLOW), which is fast enough to set up and run that it can be used to examine numerous configurations with reasonable computational resources, and that is capable of providing design engineers with acoustic predictions that include blade-vortex interaction noise (like free-vortex methods). The research in this dissertation aims to address all of these points by presenting a new approach that bridges the gap between the low-fidelity free-vortex methods and the high-grid-resolution Eulerian methods. The basic idea is a lower-grid-resolution Eulerian method coupled with new techniques to include the fuselage interaction and blade-vortex interactions at a more reasonable computational cost than the high-grid resolution methods.

1.2 Review of Noise Prediction Technique

To understand the requirements placed upon a flow solver intended for use as part of an acoustics-prediction package it is useful to review the acoustic
prediction methods used in modern rotorcraft analysis.

Rotorcraft acoustics prediction codes such as PSU-WOPWOP (Ref. 19) and PARIS (Ref. 20) are based on the principles of Lighthill’s acoustic analogy. This analogy presents a rearrangement of the Navier-Stokes and continuity equations into the form of a wave equation on the left hand side and the remaining portions of the equation on the right as source terms. Ffowcs Williams and Hawkings extended this analogy to surfaces in arbitrary motion (Ref. 21). The equation they developed is an exact reformulation of the continuity and Navier-Stokes equations using generalized functions, and can be written in the form of an inhomogeneous wave equation:

\[ \square^2 p'(\bar{x}, t) = \frac{\partial}{\partial t}\left\{[\rho_0 U_n + \rho(u_n - U_n)]\delta(f)\right\} \\
- \frac{\partial}{\partial x_i}\left\{[P_{ij}\hat{n}_j + \rho u_i(u_n - U_n)]\delta(f)\right\} \\
+ \frac{\partial^2}{\partial x_i \partial x_j}\left[T_{ij}H(f)\right] \]  

(1.1)

Note that the generalized differentiation is implicit in the first two terms due to the presence of the Dirac delta function, \(\delta(f)\). Here the Heaviside function \(H(f)\) is used to remind the reader that the volume source term is only considered outside the surface \(f = 0\), and the overbar on the wave operator \(\square^2\) reminds the reader that all derivatives in that operation are also generalized derivatives. (Refer to the List of Symbols on page xii for remaining symbol definitions).

The three terms on the right hand side of the Ffowcs Williams–Hawkings (FW–H) equation are referred to as the monopole, dipole and quadrupole terms, respectively, due to their mathematical structure. For subsonic noise generation the surface source terms (the monopole and dipole) are the most significant,
and because volume integrals are computationally expensive, the acoustics code used in this research, PSU-WOPWOP, neglects the quadrupole term. The effects of the quadrupole term can be included by integration over a permeable surface that encloses the important acoustic source regions.

To solve the FW–H equation, PSU-WOPWOP uses a time-domain integral formulation developed by Farassat (Ref. 22), Farassat’s Formulation 1A. This formulation excludes the quadrupole term of the FW–H equation and is valid for any rigid-body surface motion:

\[ p'(\vec{x}, t) = p'_T(\vec{x}, t) + p'_L(\vec{x}, t) \]  

(1.2)

where the thickness contribution \( p'_T \) is calculated from

\[
4\pi p'_T(\vec{x}, t) = \int_{f=0} \left[ \frac{\rho_0(U_n + U_R)}{r(1 - M_r)^2} \right]_\text{ret} dS \\
+ \int_{f=0} \left[ \frac{\rho_0 U_n (r M_r + c(M_r - M^2))}{r^2(1 - M_r)^3} \right]_\text{ret} dS 
\]  

(1.3)

and the loading contribution \( p'_L \) is calculated from

\[
4\pi p'_L(\vec{x}, t) = \frac{1}{c} \int_{f=0} \left[ \frac{\dot{L}_r}{r(1 - M_r)^2} \right]_\text{ret} dS \\
+ \int_{f=0} \left[ \frac{L_r - L_M}{r^2(1 - M_r)^2} \right]_\text{ret} dS \\
+ \frac{1}{c} \int_{f=0} \left[ L_r \frac{r \dot{M}_r + c(M_r - M^2)}{r^2(1 - M_r)^3} \right]_\text{ret} dS 
\]  

(1.4)
Here,

\[ U_i = \frac{\rho}{\rho_0} (u_i - v_i) + v_i \]  

(1.5)

\[ L_i = P_{ij} \hat{n}_j + \rho u_i (u_n - v_n) \]  

(1.6)

Note that in Equation 1.6 the \(-p_0 \delta_{ij}\) term has been combined into the compressive stress tensor \(P_{ij}\), so for an inviscid fluid \(P_{ij} = p' \delta_{ij}\).

In these equations, two principal types of input requirements can be identified: geometric and fluid dynamic. Appearing in all terms in Formulation 1A is the radiation vector \(r\), the vector pointing from the acoustic source location to the observer (e.g. microphone, etc.). To calculate this vector at any emission time \(\tau\) requires complete knowledge of the shape and position of the surface. In addition, to calculate terms such as \(M_r, \dot{M}_r, \) and \(U_i\), the velocity and acceleration of all points on the surface is also required.

In addition to the complete specification of the geometry and motion of the acoustic data surfaces, fluid dynamic inputs are also required. In some types of simulations (those where the acoustic data surfaces coincide with the physical body surfaces, e.g. the rotor blades), input of the scalar gage pressure may be adequate. This omits the drag contribution to the loading vector \(L\), but is frequently adequate for basic acoustics predictions. If the input is calculated from a viscous flow solver, the full loading vector may be input on the blade surface. Alternately, in the arrangement of Formulation 1A presented here, the integration surface does not need to necessarily coincide with the surface of the rotor blade—it may be placed some distance from the surface to allow the capture of high-speed impulsive noise, etc. In cases where the acoustic data surface does not correspond to a body surface, the input of the full solution vec-
tor \( Q = (\rho, u, v, w, p) \) is required at each point on the integration surface. For the acoustics presented in this research, only impermeable on-blade surfaces are presented.

Note that in addition to \( L \), Formulation 1A includes a contribution from its time derivative, \( \dot{L} \): this term is especially important for BVI noise prediction, which is highly impulsive in nature. That is, in a strong BVI situation, not only is the blade force itself critical, but its time derivative must also be adequately captured by the flow solution. Modern Eulerian solvers are often not able to capture and convect rotor tip vortex strengths well enough for the direct capture of BVI noise without very high spatial resolution of the computational domain. While these solvers excel at predicting bulk flow effects and other larger-scale interactional aerodynamics, the extremely small area and very high gradients that make up a rotor tip vortex are generally not well-captured. While the position of the vortex center is generally well-captured by these techniques (more so than with the much faster but more approximate free-vortex methods), its strength is greatly under-predicted. If an Eulerian solver is to be used as an input to an acoustic prediction code, some additional technique must be used to capture BVI noise. One significant contribution of the present research is the development of a new approximate technique for modeling the effect of BVI on the acoustics without directly capturing the impulsive loading in the flow solution procedure.
1.3 Rotorcraft Noise Sources

A rotorcraft has numerous interdependent noise-generation mechanisms. Depending on the details of the configuration and on the flight condition of interest, some or all of these may be important for noise prediction:

**Blade motion.** In the Ffowcs Williams-Hawkings (FW–H) equation (see Section 1.2), the noise is divided into three sources: monopole, dipole and quadrupole. When the integration surface \( f = 0 \) is taken to be the blade surface, the monopole term contains no fluid-dynamic terms. This parameter is called “thickness noise,” and depends only on the shape of the blade and its motions. Due to the distribution and motion of these monopole sources on the rotor blades, thickness noise is primarily directed in the plane of the rotor, so its prediction is especially critical for rotors aligned perpendicular to the ground, for example a tail rotor or pusher-prop, or for observers far ahead of the rotor, in acoustic detection and stealth problems.

**Blade loading.** The dipole term in the FW–H equation is referred to as the “loading noise” term when \( f = 0 \) is the blade surface. This term is dependent on the surface pressure distribution on the blade. Anything that affects the blade loading will affect this term, i.e. any kind of rotor inflow distortion or wake interaction. For example, in forward flight the forward section of the fuselage directs flow upwards through the front section of the rotor disk: this causes an inflow asymmetry, and therefore a once-per-rotor-revolution (1/rev) influence on the blade loading. The closer the rotor blades pass to the fuselage, the greater the effect. Most mod-
ern helicopters (for example, the new Bell 429 in Fig. 1.1a) are designed to minimize this distance in order to decrease the hub drag, so acoustic analysis of the main rotor in forward flight is dependent on this interactional aerodynamic phenomenon. A tiltrotor aircraft such as the V-22 Osprey (Fig. 1.1b) or the proposed heavy-lift Quad Tilt Rotor (QTR, Fig. 1.1c) have an additional inflow distortion due to the presence of the wing. In hover, for example, this distortion is a complicated, unsteady “fountain flow” region that oscillates between the rotors.

**Blade vortex interaction.** The loading on a blade is also strongly affected by the presence of the tip vortices shed from the rotor blades. In a conventional helicopter this wake can interact with the main rotor (in forward or descending flight) and with the tail rotor (in forward flight), causing significant impulsive noise. In coaxial rotor designs such as the Sikorsky X2 shown in Fig. 1.1d, the flight regimes in which this impulsive noise source is important multiply, making accurate prediction of the tip vortices essential for all flight conditions. In addition, due to their larger number of rotors, some heavy lift configurations have increased the opportunity for blade-vortex interactions. Many of these configurations place one or more rotors directly downstream of the others. The prediction of blade-vortex interaction (BVI) noise requires a flow solver that can accurately predict and convect the vortices for relatively long periods of time (tens of rotor revolutions for tail rotor noise, for example).

**Shock formation.** In advanced high-speed helicopters the tip Mach number can be quite high in forward flight, sometimes exceeding Mach 0.9. Local
accelerated flow regions can be supersonic, causing shockwaves to form. If these supersonic regions extend off the blade into the farfield (delocalization), there is a dramatic increase in rotor noise (Ref. 23). Prediction of this high-speed impulsive (HSI) noise requires a compressible flow solver, and the noise cannot be computed from the blade surface pressure, as the HSI noise is generated by a volume source.

**Other interactions.** Many other aerodynamic phenomena affect the noise generated by a helicopter, though they are not always important for early design decisions. Interaction with the turbulent wake generated by preceding blades, the separated flow region behind a fuselage or the rotor hub, etc. can all affect the noise. Prediction of these effects requires a highly sophisticated flow solver with good turbulence modeling or prediction capability.
Figure 1.1: Advanced rotorcraft configurations.
1.4 Summary of Innovations

This dissertation presents a number of related innovations, starting from its fundamental goal of providing faster noise calculations over complex heavy-lift rotorcraft. The design of the flow solver has been driven entirely by this unique requirement. In particular, these innovations include:

- A new immersed boundary technique to support inviscid flow solutions on very coarse grids.
- A new technique for capturing the acoustic effects of blade-vortex interactions.
- An end-to-end acoustic prediction system designed to reduce the cost of acoustics predictions by minimizing both computer time and engineer time.

These items taken together represent a substantial advancement in the state of the art for acoustic predictions in the design phase of a rotorcraft, giving engineers the ability to quickly and easily set up and run new cases with advanced configurations, without requiring manual grid generation or high-resolution meshes. Future work should include efficiency enhancements that will reduce the turnaround time and make this tool truly viable for the development of efficient low-noise rotorcraft.
Chapter 2

Literature Review

This literature review is divided into two sections: a review of Immersed Boundary CFD methods, and a review of BVI prediction methods.

2.1 Immersed Boundary CFD Methods

Immersed Boundary (IB) methods take many forms: some literature makes a distinction between “immersed” and “embedded” boundaries (e.g. Ref. 24), while others treat them as extensions of the same concept (e.g. Ref. 11). In the most general definition presented by Mittal and Iaccarino in Ref. 11, an immersed boundary technique is any flow solution technique where the grid does not conform to the surface of the geometry that the fluid is flowing around. This can include structured and unstructured approaches (though the vast majority are structured), finite difference such as Ref. 12 and finite volume methods such as Ref. 25, rigid or deformable body boundaries (Ref. 26 vs. 27, for example), and can involve the solution of any of several fluid equations of motion: compressible (e.g. Ref. 28) or incompressible (e.g. Ref. 29), viscous (e.g. Ref. 30) or
inviscid (e.g. Ref. 13), as well as the potential inclusion of separate equations of motion governing the location of the boundary itself (as in a flexible membrane case, e.g. Ref. 31).

Finite Volume approaches are sometimes referred to as “Cartesian Grid Methods” or “Cut Cell” methods in the literature, e.g. Refs. 25,32–35. The basic premise of a cut cell method is that the intersections between grid cells and the geometry are calculated, and the resulting “cut cells” are used to represent the body surface. This yields a body-fitted surface treatment even though the original grid did not coincide with the geometry surface. One of the major challenges in implementing a cut-cell method is the calculation and proper handling of the flow cells that intersect, or are cut, by the body (see, e.g. Ref. 36). In addition to the difficulties in tracking the faces of such cells (which may be very complex), in general the timestep restriction for the solver is driven by the volume of the smallest cell. Considerable research has been performed in merging these small cells with their larger neighbors to alleviate this potential problem (see, e.g. Refs. 37 and 25). Naturally, this also increases the problem complexity. Most techniques for implementing finite volume cut-cell solvers also require a relatively “clean” geometry, and many require a wetted surface input (that is, is the case where parts intersect, the intersection must be calculated and trimmed such that no part of the geometry exists inside any other part). Cleaning up direct-from-CAD geometry is a complicated and expensive problem, involving substantial ongoing research (see e.g. Ref. 36). The principal advantage of these techniques is that they allow direct application of the surface non-penetration boundary condition at the cut cell faces.

Finite-difference “Immersed Boundary Methods” stem from Peskin’s 1972
paper “Flow Patterns Around Heart Valves: A Numerical Method,” (Ref. 38), in which he solves the fluid-structure interactions of flow through a heart by replacing the physical boundaries with a field of force acting on a set of mesh points in a rectangular domain. In this finite-difference, viscous, incompressible technique, the force field is dynamically calculated at each timestep and the walls of the heart are allowed to move in response to the fluid pressure acting on them. Key to his method is his “semidiscrete” representation of the Dirac $\delta$ function used to represent the immersed boundary. Peskin spreads the force acting at the surface of the body across several cells using a distribution function to approximate the Dirac $\delta$ function. The use of some distribution function is a key difference between the work of later researchers, some of whom continue to use a distributed Dirac $\delta$ function and some of whom treat the body as a sharp interface using a variety of methods.

The literature stemming from Peskin’s work can be divided into two broad categories: direct and indirect boundary condition application techniques. Indirect boundary condition applications techniques attempt to modify the flow at points interior to the body such that when a derivative is taken near the body surface, the correct flux is calculated at or near the surface location. These techniques involve calculating a forcing term that is added to the right-hand side of the momentum equations, and differ in the manner of calculating this “penalization” force. In these penalization techniques, the calculation of the fluxes is not directly influenced by whether a gridpoint is interior or exterior to the body. Instead, at each timestep a “penalization force” is calculated to bring the fluid at gridpoints that lie inside the body up to a specified velocity. The actual procedure for taking the flowfield derivatives proceeds as though no body were
present, and the presence of the body is implicitly simulated through the action
of the calculated force. A penalization method can be further subdivided into
two sub-categories: distributed-forcing and discrete forcing. In Peskin’s origi-
nal formulation the force acting on the walls of the heart was distributed over
several cells in order to stabilize the method. Various representations for this
force distribution are investigated in the literature.

In these distributed forcing techniques, rigid bodies are challenging to
model: the equations of motion developed by researchers above are not well-
posed in the inelastic limit, so various methods have been developed to use
these techniques for simulating the flow over rigid bodies. Beyer and Leveque
(Refs. 39, 40) present a technique for treating the body as connected to a spring
with some pre-set restoring force. Unfortunately, accurately simulating a rigid
body requires this restoring string constant to be very high, resulting in a very
stiff numerical system that requires very small timesteps to solve. Lai and
Peskin (Ref. 41) present a similar method that allows the surface to undergo
small motions that is formally second-order accurate. Leveque et al. (Ref. 33)
presents an alternate method based on a combination of finite-difference and
finite-volume techniques. Goldstein et al. (Ref. 42) developed a more general
version of the same concept where the effect of the rigid body is imposed on the
surrounding flow by a damped harmonic oscillator-type feedback mechanism
intended to actively control the velocity near the surface. Unfortunately, this
type of technique requires the choice of two coefficients and is again subject
to severe timestep limitations, especially for unsteady and/or high-Reynolds
number flows. Saiki (Ref. 43) demonstrated the use of this technique for flow
over rigid cylinders, distributing the force applied over a range of nearby cells
via a Gaussian distribution. Angot et al. (Ref. 44) and Khadra et al. (Ref. 45) developed an alternate technique based on Brinkman’s porous media equations (Ref. 46) that assumes the entire flow occurs in a porous medium: the porosity of the points interior to the body are then assumed to be very low. In practical implementation terms this technique is very similar to the other distributed forcing techniques and results in relatively poor enforcement of the surface boundary conditions. It has been demonstrated to give good results for flow over simple geometries at very low Reynolds numbers, however (Ref. 45).

For more accurate treatment of rigid bodies such as those more conventionally seen in aerospace applications, another technique is needed. The alternative to the distributed forcing penalization method is the discrete-forcing penalization method. Mohd-Yusof and Verzicco (Refs. 47–50) developed this new method for calculating the force field for a stationary (non-interacting) boundary. In this type of technique a forcing function is added to the discretized Navier-Stokes equations to simulate the effect of the body of the flow. Fadlun et al. (Ref. 51) presents an analysis of interpolation methods for the calculation and application of the surface boundary conditions in penalization methods. In particular they look at the various techniques used to calculate the forcing function, ranging from a zeroth-order “interpolation” from the original Mohd-Yusof implementation, up to a second-order velocity interpolation technique that attempts to more accurately force the boundary conditions at the surface location. In the zeroth-order methods, the force is applied such that the flow is brought to rest at the first gridpoint that lies within the solid body, with no regard given to the actual location of the body surface in between the treated gridpoints. Higher-order methods attempt to calculate a forcing function that
takes into account the true location of the immersed boundary. Fadlun et al. demonstrate substantial improvement to the immersed boundary calculations when using the higher-order surface treatments.

In contrast to these “penalization” methods, a reconstruction method (or a “direct boundary condition application method” in some of the literature) modifies the calculation of the fluxes at the body surface to account for the presence of the body. These techniques can take a number of different forms: in a “ghost cell” technique, derivatives are taken across the body surface, but the points interior to the body are not used, or are used merely as convenient storage locations. Instead, the derivative stencil makes use of “ghost” cells whose values are calculated to yield a specified value for the flux at the body surface. In a “surface reconstruction” technique no derivatives are taken at the layer of cells nearest the body. Instead, these points are explicitly set via a boundary condition application.

Majumdar et al. (Ref. 52) present a ghost-cell-type method for treating the immersed boundary conditions and demonstrate several additional interpolation techniques. Dadone and Grossman (Ref. 13) extend this ghost cell technique using a “curvature corrected symmetry technique (CCST)” using the normal momentum equation and assumed model flow to better represent the flow near the immersed boundary. In 2003 Gilmanov et al. (Refs. 27, 53) presented a similar technique, treating the boundary as a sharp interface and using the normal momentum equation and extrapolation from the flowfield to reconstruct the boundary conditions at the body surface.

Flow simulations have been primarily incompressible, but to a limited extent compressible flows have been examined as well. Numerous researchers
have examined alternate methods for treating the no-slip wall boundaries, some including local grid refinement, non-Cartesian grids, etc. A wide variety of various related techniques have been studied, and if recent publications are any indication, interest in the technique appears to be increasing (see e.g. Refs. 12, 26, 30, 31, 54–70). In addition, Griffith et al. have examined the convergence rates of the distributed force schemes (Ref. 71), and Udaykumar et al. (Ref. 34) present a method for dealing with a rigid body moving through a stationary grid, and develop a technique for dealing with “freshly-cleared” cells, i.e. those that were interior on the previous timestep and are exterior on the current timestep, or vice versa.

The vast majority of past work has focused on viscous flows [in his 2005 review (Ref. 11) Mittal explicitly limits the term “immersed boundary methods” to viscous flows]. Only Dadone and Grossman have shown inviscid slip-wall flow results: 2D in Refs. 13, 72 and some limited work on 3D in Refs. 35, 73. In their “ghost-cell technique” they use a non-cut-cell finite volume technique and a surface reconstruction similar to Gilmanov’s. They model the flow at the surface using an assumed vortex sheet and have shown very good grid convergence results, and good stability on fine grids.

Finally, several review articles have been published. Peskin provides a good review of δ-type methods in Ref. 74 and Mittal and Iaccarino present an excellent review of the 2005 state of the art in Ref. 11.
2.2 Literature Review of Vorticity Preservation Techniques

In rotorcraft, the strong, discrete tip vortices generated at the outboard rotor tips can interact with following blades, generating a large impulsive loading on the blades, resulting in very high noise levels. The computation of these blade-vortex interactions depends on the ability of the flow solver not only to generate the tip vortices, but to convect the vortices without non-physical dissipation over very long computation times. Predicting BVI noise has long been an important research topic in the rotorcraft noise field. In a conventional helicopter it is particularly important for community noise, as the frequency range falls into that in which the human ear is most sensitive. The most important source of BVI noise in a conventional helicopter is in general generated at the advancing side of the main rotor at azimuths of 70–80 degrees, where a vortex shed by a previous blade passage may interact with a successive blade in a nearly parallel manner. This type of BVI propagates downward and forward at an angle of approximately 30–40 degrees to the rotor tip-path plane (Ref. 75). This type of BVI occurs on the helicopter main rotor in descending forward flight, for example on approach to a landing zone.

Modern rotorcraft, and especially heavy-lift rotorcraft, pose a more complicated problem: many current heavy-lift designs involve multiple main rotors, pusher props, and other non-conventional rotor layouts. These result in not only increased opportunity for BVI due to the possibility of a rotor interacting with the vortices generated by itself and the other rotors (depending on the configuration), but in some designs the rotor tip path plane is sometimes tilted,
resulting in a directional change in the BVI produced. This has not been well-studied, and it is not known what the impact of BVI on these designs will be: in particular, while conventional BVI is considered a community noise problem, it is in general not directed far forward of the helicopter, and is thus not important from the standpoint of military aircraft detectability. The influence on detectability of the differing interaction locations and tip-path-plane angles of a heavy-lift rotorcraft is not known at this time. It is thus important for both military and civil heavy-lift rotorcraft that any tool for analyzing their noise include the capability of predicting BVI noise, at least in an approximate way.

2.2.1 Lagrangian Solvers

In 1962 Piziali (Ref. 76) presented an early Lagrangian method for calculating the airloads distribution in forward flight which included the presence of the tip vortices. Then in the early 1970s Widnall (Ref. 77) presented one of the earliest theoretical studies of rotor BVI noise: using linear unsteady aerodynamic theory and an oblique gust model, she was able to predict the basic effects of 2D BVI noise quite well. Summa (Ref. 78) then performed 3D potential flow calculations applied to rotors including the shedding and roll-up of the rotor tip vortex. Later theoretical work on vortex noise generation includes an analysis of BVI in low Mach number flow by Obermeier (Ref. 79).

The next step beyond simple panel methods and prescribed wake geometries is to implement a method for predicting the flowfield that explicitly captures the motion of the rotor tip vortices from first principles. Lagrangian solvers such as HELIX-I and HELIX-II (Ref. 80), the free-vortex module in CAMRAD and its derivatives (Ref. 81, 82), the Maryland Free Wake Code (Ref. 83),
and many other free-vortex implementations solve for the shed vortex locations directly by integrating the Biot-Savart integral for a set of vortex filaments and allowing the vortices to move under the influence of the induced velocity field. Considerable progress has been made towards developing fast, efficient, scalable solvers using this technique (See Ref. 84 for a review). These solvers can include the influence of the fuselage in a potential-flow sense, but are generally unable to include viscous or compressibility effects except through approximate models. An extensive literature exists for these types of solvers, and considerable active research is underway refining and studying the models. In particular, these types of solvers have been used to predict BVI noise in Refs. 85–87.

There are several disadvantages to these free-wake solvers, however. First, because most are limited to potential flow solutions, they are unable to capture any viscous effects (though some viscous particle methods do exist, for example Ref. 88). Their efficacy in predicting noise in complex highly-interactional flows is unknown, as they are most often used in isolated rotor simulations. When applied to even relatively “simple” interactional problems such as tail rotor noise (not a simple problem by any means, but straightforward compared to some of the highly-complex heavy-lift concepts being proposed) they have been mostly unsuccessful in yielding useful results (see e.g. Refs. 89–92). In addition, most are not implemented using parallel computational methods, so despite their low computational complexity relative to Eulerian flow solvers, they still consume significant wall-clock computational time in addition to the human time required to set up the case. Finally, they are unable to predict compressibility effects except through coarse models, and cannot be used to predict
high-speed impulsive noise, a potentially important noise source in high-speed forward flight.

2.2.2 Navier-Stokes Eulerian Solvers

Moving beyond the relatively low-fidelity flow solutions of most Lagrangian flow solvers, Navier-Stokes solvers are the workhorse of the CFD community, but pose significant challenges to the adequate capture and convection of vorticity. In the late 1980s, Srinivasan presented some early 2D computations of an Eulerian blade-vortex interaction (Ref. 93). In his 1989 Ph.D. thesis, Baeder also presents computational studies of 2D blade-vortex interactions using an Eulerian rather than Lagrangian approach (Ref. 94). Hassan et al. used CAMRAD to insert a 3D vortex into a flowfield that was then convected using an Euler code in Ref. 95. Baeder and Srinivasan then used a modified version of the Transonic Unsteady Rotor Navier-Stokes (TURNS) CFD code to predict blade surface pressures and nearfield acoustic predictions, for the first time demonstrating the potential for using Eulerian CFD techniques to predict rotor BVI noise directly (Refs. 75, 96–98). Unfortunately, due to numerical discretization errors, most Eulerian flow solvers dissipate the rotor tip vortices much faster than the physical vortices dissipate, and acoustic calculations clearly show the limitations of using a standard Navier-Stokes solver to calculate the influence of vortex interactions on the noise (e.g. Ref. 99).

Researchers have attempted to overcome this difficulty using a variety of techniques. In Refs. 100–102, researchers apply techniques for the dynamic creation of structured Chimera (overset) grids that follow the tip vortex trajectory. Recent research into advanced vorticity-preserving differentiation schemes in
otherwise standard Navier-Stokes solvers shows considerable promise when used with high grid densities (Ref. 103). Unfortunately, these schemes are computationally very expensive, and still require very large numbers of grid cells in order to prevent non-physical dissipation of the vortices.

2.2.2.1 Vorticity Confinement

John Steinhoff pioneered a technique for preventing the artificial numerical dissipation of vortices by adding a forcing term to the equations that is related to the strength of the vortex and the grid spacing (which influences the rate of artificial dissipation) (Refs. 104–107). This technique, termed “vorticity confinement,” is relatively simple to implement in existing CFD codes, but requires at least one additional parameter to be specified: essentially, the length and time scale at which the vorticity will be confined. This parameter is generally defined such that it depends on the size of the grid cells in the vortical region, allowing vorticity to be convected directly where the grid is fine enough to do so, and preserving the vortex strength in regions where the grid becomes too coarse. Some research has been done using a simpler definition, but the results showed significant numerical instabilities for certain choices of length scale (Ref. 108). In addition, although the original formulation was applied to the unsteady Euler equations, other researchers have successfully applied it to the compressible Navier-Stokes equations (Ref. 109). In general, however, the technique has proven difficult to apply to higher order methods. See Ref. 110 for further details and analysis of this technique.
2.2.2.2 Vorticity Transport Solvers

Another possible resolution to the vorticity dissipation problem in Eulerian solvers is to cast the equations in a vorticity-conservation form, rather than a momentum-conservation form as in the Navier-Stokes equations. Brown and others have demonstrated the solution of this vorticity transport equation on an Eulerian mesh. When coupled with an appropriate flux-limiting scheme this technique is able to preserve the strength of the vortices through very long convection times (Refs. 15, 17). In recent years the scheme has been applied to various rotorcraft acoustics problems (including some heavy-lift configurations: see e.g. Ref. 111). This is a very promising technique for predicting the complex interactional aerodynamics flows—unfortunately, it still requires the development of a body-fitted grid system (requiring a clean geometry input and substantial human time investment). In addition, the technique is incompressible by construction, which limits its fidelity.

2.2.3 Coupled Lagrangian/Eulerian Techniques

In addition to the isolated approaches, several researchers have studied the use of combine Lagrangian/Eulerian solutions for the prediction of helicopter BVI. The general concept is to use a compressible, viscous Navier-Stokes flow solver in the region nearest the blade where the flow must be modeled as compressible, but to then switch to using a Lagrangian particle-based method in the region away from the blade, where compressibility and viscous effects are less important (see e.g. Refs. 14, 80, 112). To date, none of these techniques has been demonstrated for acoustic predictions, and there is still considerable ongoing
research into the precise manner of handling the coupling of the fluid equations of motion at the compressible/incompressible boundary.
Chapter 3

Immersed Boundary Fuselage Representation

The goal of the research in this dissertation is the development of techniques that enable a flow solver to be set up quickly to run a variety of geometries, as might be used during the design of a new rotorcraft. To achieve low setup costs, a solver based in part on the Immersed Boundary (IB) method was developed. For this research the immersed boundary technique was used within the framework of an otherwise-conventional compressible, inviscid flow solver, using body-fitted grids to model the rotor blades (which are relatively simple to grid) and using immersed boundaries to include the bulk effects of the fuselage in the flow solution. In addition, the flow solver design is heavily influenced by the main project goal of using the solver to quickly predict the noise generated by complex configurations. This influence is reflected in the name of the new solver: the Immersed Boundary Solver for Environment Noise (IBSEN).
3.1 Flow Solver Overview

The governing equations of motion being solved in this research are the inviscid, compressible Euler equations of fluid dynamics (see Section 3.2). The compressible equations of motion are used because the rotor blades typically operate in the high-subsonic and transonic flow regimes, where compressibility is important. Viscosity is neglected as a second-order effect for the purposes of this research, where the aim is to capture the basic acoustic signature of the aircraft, neglecting finer-scale detail that a more advanced viscous solver might achieve, instead providing results faster and with less computational effort.

To capture the complex flowfield around the rotor itself a system of conventional structured, curvilinear, overset, body-fitted meshes is used. This is embedded within an off-body grid system consisting of successive layers of uniform Cartesian meshes, with each layer becoming progressively coarser as they move away from the rotor. The overall case setup is similar to that used with other flow solvers such as OVERFLOW. However, unlike most other solvers of this type, the fuselage geometry is accounted for without developing a body-fitted grid system. Instead, the fuselage is simply immersed in the same grid used to model the isolated rotor. Since the actual immersed boundary calculations are inexpensive, this results in a flow solver that can model the effects of the fuselage on the rotor aerodynamics for effectively the same computational cost as the isolated rotor system.

Because the focus of this research is on inexpensive acoustic calculations, a simple second-order in space, first-order in time, implicit Beam-Warming flow solver is used. This type of flow solver is robust and fast, trading solution accu-
racy for speed and simplicity. While the techniques developed in this research are only shown in the context of this simple flow solution technique, it is expected that for the most part they will also extend without modification to more advanced timestepping and differentiation schemes.

Finally, once the aerodynamic forces on the rotor blades are known, an acoustic propagation code such as PSU-WOPWOP is used to calculate the noise at the observer location (or locations). IBSEN generates all of the necessary input files for PSU-WOPWOP, and will call it directly if it is available on the system where IBSEN is run.

### 3.2 Governing Equations

For this research the compressible, inviscid Euler equations are written for an arbitrary moving coordinate system; the flow solution is ultimately computed for and stored in the inertial frame. The implementation is based on Pulliam and Steger’s formulation (Ref. 113 and Ref. 114):

\[
\partial_t \hat{q} + \partial_\xi \hat{E} + \partial_\eta \hat{F} + \partial_\zeta \hat{G} = 0
\]  

(3.1)

where

\[
\hat{q} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ e \end{bmatrix} \quad \hat{E} = \begin{bmatrix} \rho \\ \rho u U + \xi_x p \\ \rho v U + \xi_y p \\ \rho w U + \xi_z p \\ (e + p) U - \xi_i p \end{bmatrix}
\]
\[ \hat{F} = \begin{bmatrix} \rho \\ \rho uV + \eta_x p \\ \rho vV + \eta_y p \\ \rho wV + \eta_z p \\ (e + p)V - \eta_t p \end{bmatrix} \]

\[ \hat{G} = \begin{bmatrix} \rho \\ \rho uW + \zeta_x p \\ \rho vW + \zeta_y p \\ \rho wW + \zeta_z p \\ (e + p)W - \zeta_t p \end{bmatrix} \]  \tag{3.2} 

and

\[ U = \xi_t + \xi_x u + \xi_y v + \xi_z w \]

\[ V = \eta_t + \eta_x u + \eta_y v + \eta_z w \]

\[ W = \zeta_t + \zeta_x u + \zeta_y v + \zeta_z w \]  \tag{3.3} 

\( U, V, \) and \( W \) are the contravariant velocities, as in Ref. 113 and . \((\xi_t, \eta_t, \zeta_t)\) is the grid velocity and the matrix

\[ \begin{bmatrix} \xi_x & \xi_y & \xi_z \\ \eta_x & \eta_y & \eta_z \\ \zeta_x & \zeta_y & \zeta_z \end{bmatrix} \]  \tag{3.4} 

is a combination of the Jacobian matrix and linear transformation matrix (i.e. the rigid-body rotation and translation of the grid at the current timestep).

### 3.2.1 Explicit Method

For basic research purposes simple explicit second- and fourth-order Runge-Kutta methods were implemented (Ref. 115). For spatial discretization in the explicit cases, first- and third-order upwind schemes as well as a fifth-order
Weighted Essentially Non-Oscillatory (WENO) scheme are currently implemented with Lax-Friedrichs flux splitting. Because of timestep size limitations on implicit methods when used with immersed boundary methods (Ref. 116) it is often more efficient to use a high-speed explicit method when no body-fitted curvilinear meshes are present and only the immersed boundary is being considered. For example: when using the first-order time, second-order spatial accuracy implicit Beam-Warming method discussed in the next section, the maximum Courant number for the run is $\nu = 1$, even though a conventional implicit flow solution method has no theoretical CFL limit—the immersed boundaries prevent larger timesteps from being used, greatly limiting the utility of the method.

For enhanced solution stability in the explicit method a sixth-order implicit filter is used to remove spurious oscillations generated at the overset and body boundaries. The filter chosen is that of Visbal and Gaitonde (Ref. 117):

$$
\alpha_f \hat{\phi}_{i-1} + \hat{\phi}_i + \alpha_f \hat{\phi}_{i+1} = \sum_{n=0}^{N} \frac{a_n}{2} \left( \phi_{i+n} + \phi_{i-n} \right) 
$$

where $\phi$ are the original values and $\hat{\phi}$ the filtered values. The coefficients used can be found in Table IV of Ref. 117 and the frequency response in Ref. 118. In general this low-pass filter serves to remove frequencies at or above the maximum-supportable frequency of the grid (based on the Nyquist criterion). The tridiagonal system is solved using the LAPACK library routines, so this segment of code can benefit from the use of high-performance system libraries such as those developed by Intel for their architectures. The filter requires approximately 15% of the total code runtime as implemented using non-vendor-
optimized reference libraries.

### 3.2.2 Implicit Method

For more advanced cases, where the presence of a curvilinear surface mesh (and its correspondingly small surface cells) prevent the practical application of an explicit method, IBSEN also implements the Euler implicit first-order in time approximate factorization algorithm of Beam and Warming (Ref. 119). While this implicit method still suffers from the CFL limitation on any immersed boundaries, for rotor cases where the blades are modeled using curvilinear surface meshes and only the fuselage is modeled using immersed boundaries, the flow speed is low enough and the grid cells large enough near the immersed boundaries that the local CFL requirement in those regions is met by conventional rotor analysis step sizes (on the order of a tenth of a degree or rotor revolution per timestep).

The approximate factorization used is

$$
\left( I + \Delta t \delta_\zeta \hat{A}^n \right) \left( I + \Delta t \delta_\eta \hat{B}^n \right) \left( I + \Delta t \delta_\zeta \hat{C}^n \right) \left( \hat{q}^{n+1} - \hat{q}^n \right) \delta_\zeta = - \Delta t \left( \delta_\zeta \hat{E}^n + \delta_\eta \hat{F}^n + \delta_\zeta \hat{G}^n \right) \tag{3.6}
$$

Here the $\delta$’s are second-order central-difference operators and the Jacobian matrices $\hat{A}^n$, $\hat{B}^n$, and $\hat{C}^n$ are obtained from the time linearization of $\hat{E}^n$, $\hat{F}^n$, and $\hat{G}^n$: 
\[ \hat{A}^n, \hat{B}^n, \text{or} \hat{C}^n = \]
\[
\begin{bmatrix}
K_0 & K_1 & K_2 \\
K_1\phi^2 - u\theta & K_0 + \theta - K_1(\gamma - 2)u & K_2u - (\gamma - 1)K_1v \\
K_2\phi^2 - v\theta & K_1v - K_2(\gamma - 1)u & K_0 + \theta - K_2(\gamma - 2)v \\
K_3\phi^2 - w\theta & K_1w - K_3(\gamma - 1)u & K_2w - K_3(\gamma - 1)v \\
\theta[2\phi^2 - \gamma(e/\rho)] & K_1Y - (\gamma - 1)u\theta & K_2Y - (\gamma - 1)v\theta \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
K_3 & 0 \\
K_3u - (\gamma - 1)K_1w & K_1(\gamma - 1) \\
K_3v - (\gamma - 1)K_2w & K_2(\gamma - 1) \\
K_0 + \theta - K_3(\gamma - 2)w & K_3(\gamma - 1) \\
K_3Y - (\gamma - 1)w\theta & K_0 + \gamma\theta \\
\end{bmatrix}
\]

where

\[
\phi^2 = 0.5(\gamma - 1)(u^2 + v^2 + w^2)
\]

(3.8)

\[
\theta = K_1u + K_2v + K_3w
\]

(3.9)

\[
Y = \gamma(e\rho) - \phi^2
\]

(3.10)

and, to obtain \(\hat{A}\),

\[
K_0 = \xi_t, \quad K_1 = \xi_x, \quad K_2 = \xi_y, \quad K_3 = \xi_z
\]

(3.11)

Likewise, to obtain \(\hat{B}\),

\[
K_0 = \eta_t, \quad K_1 = \eta_x, \quad K_2 = \eta_y, \quad K_3 = \eta_z
\]

(3.12)
and $\hat{C}$,

$$K_0 = \zeta_t, \quad K_1 = \zeta_x, \quad K_2 = \zeta_y, \quad K_3 = \zeta_z$$

(3.13)

### 3.2.3 Artificial Viscosity

To stabilize the implicit equations, ARC3D-type artificial viscosity is included (Ref. 120). Some care must be taken in the implementation of these terms to ensure that flow regions that are blanked out either by oversetting or the inclusion of a body geometry are properly excluded. To do this, the notion of “iBlanking” is introduced, where an additional integer value $IB$ is evaluated at each grid location: this value is set to one at all valid mesh positions, and to zero at invalid, or “blanked” positions. First defining the following averaging and differencing formulas for arbitrary $x$:

$$\bar{x}_{k+\frac{1}{2}} = \frac{x_k + x_{k+1}}{2}$$

$$\bar{x}_k = \frac{x_{k-1} + 2x_k + x_{k+1}}{4}$$

$$\delta x_{k+\frac{1}{2}} = x_{k+1} - x_k$$

$$\delta x_k = \frac{k_{k+1} - x_{k-1}}{2}$$

$$\delta^2 x_k = x_{k+1} - 2x_k + x_{k-1}$$

(3.14)

and the following metric scaling terms:

$$k_i = \sqrt{\zeta_x^2 + \zeta_y^2 + \zeta_z^2}$$

$$k_j = \sqrt{\eta_x^2 + \eta_y^2 + \eta_z^2}$$

$$k_k = \sqrt{\zeta_x^2 + \zeta_y^2 + \zeta_z^2}$$

(3.15)
The $IB$ blanking term is evaluated by holding other indices constant:

$$IB_i = IB_{(i,j,k)}$$  \hspace{1cm} (3.16)

A spectral radius is calculated independently in each grid direction from

$$\rho_i = \frac{|U| + k_i c}{J}$$  \hspace{1cm} (3.17)

where $U$ is replaced by $V$ and $W$ for the $j$ and $k$ directions, respectively, and $J$ is the determinant of the Jacobian matrix. Further defining

$$DD_i = \frac{1}{4} \left( IB_{i-1} IB_i IB_{i+1} \right) \left| \frac{\delta^2 p_i}{\ddot{p}_i} \right|$$  \hspace{1cm} (3.18)

(where $p$ is the local pressure) and

$$Q2_i = \left( IB_{i-1} IB_i IB_{i+1} \right) \delta^2 \ddot{q}_i$$  \hspace{1cm} (3.19)

we can then define

$$DD_{i+\frac{1}{2}} = \frac{DD_i + DD_{i+\frac{1}{2}}}{\max \{(IB_i + IB_{i+1}), 1\}}$$  \hspace{1cm} (3.20)

Smoothing coefficients are then calculated from
\[ \text{COEF2}_{i + \frac{1}{2}} = 2\epsilon_2 \bar{p}_{i + \frac{1}{2}} \bar{D}D_{i + \frac{1}{2}} \]  
(3.21)

\[ \text{COEF4}_{i + \frac{1}{2}} = 2\epsilon_4 \bar{p}_{i + \frac{1}{2}} \left(1 - \min \left\{ 1, \frac{\epsilon_2}{\epsilon_4} \bar{D}D_{i + \frac{1}{2}} \right\} \right) \]  
(3.22)

\[ \text{smooth}_i = -\left( \text{COEF2}_{i + \frac{1}{2}} \delta \tilde{q}_{i + \frac{1}{2}} - \text{COEF2}_{i - \frac{1}{2}} \delta \tilde{q}_{i - \frac{1}{2}} \right) 
+ \left( \text{COEF4}_{i + \frac{1}{2}} \delta Q^2_{i + \frac{1}{2}} - \text{COEF4}_{i - \frac{1}{2}} \delta Q^2_{i - \frac{1}{2}} \right) \]  
(3.23)

where \( \epsilon_2 \) and \( \epsilon_4 \) and the user-input second- and fourth-order smoothing coefficients, respectively. \( \text{smooth}_i \) is added to the right hand side, and the remaining smoothing coefficients are then used to modify the \( \tilde{A}_i^n \), \( \tilde{B}_i^n \), and \( \tilde{C}_i^n \) matrices:

\[ \text{COEF2}_{\text{left}, i + \frac{1}{2}} = \text{COEF2}_{i + \frac{1}{2}} + 2\text{COEF4}_{i + \frac{1}{2}} \]  
(3.24)

\[ \tilde{A}_i = \tilde{A}_i - \text{COEF2}_{\text{left}, i - \frac{1}{2}} I \]  
(3.25)

\[ \tilde{B}_i = \tilde{B}_i + \left( \text{COEF2}_{\text{left}, i - \frac{1}{2}} + \text{COEF2}_{\text{left}, i + \frac{1}{2}} \right) I \]  
(3.26)

\[ \tilde{C}_i = \tilde{C}_i - \text{COEF2}_{\text{left}, i + \frac{1}{2}} I \]  
(3.27)

### 3.2.4 Comparison to OVERFLOW

IBSEN is designed to be similar to NASA’s overset Navier-Stokes flow solver OVERFLOW in most respects. It gives nearly identical results to an OVERFLOW run with the same grid and the parameters specified in Table 3.1. Small differences between OVERFLOW and IBSEN results are attributable to the precise order of operations, which differs between the flow solvers. The principal differences between IBSEN and OVERFLOW are:
• OVERFLOW is designed for speed and IBSEN for extensibility, so the basic design and layout of the codes is quite different (including being written in different languages).

• IBSEN directly computes inputs to PSU-WOPWOP and is capable of running PSU-WOPWOP during the flow solution procedure, while solvers like OVERFLOW require significant data conversion, especially for body surface pressure and compact patch acoustic inputs, and the acoustic code must be run once the OVERFLOW run is complete, as a post-processing step.

• In addition to supporting body-fitted grids, IBSEN supports immersed boundary representations of surfaces that don’t require detailed surface pressure predictions (see Section 3.7).

• IBSEN supports a new particle-tracking method for analyzing rotor blade-vortex interaction noise (see Chapter 4).

• IBSEN does not currently implement any kind of viscosity/turbulence model, higher-order space or time discretization, or many other of the myriad advanced features available in OVERFLOW.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NQT</td>
<td>1</td>
<td>No turbulence model</td>
</tr>
<tr>
<td>NITNWT</td>
<td>0</td>
<td>No subiteration</td>
</tr>
<tr>
<td>LFRINGE</td>
<td>2</td>
<td>Double-fringing</td>
</tr>
<tr>
<td>IRHS</td>
<td>0</td>
<td>Central-difference Euler terms</td>
</tr>
<tr>
<td>ILHS</td>
<td>0</td>
<td>Beam-Warming block tridiagonal</td>
</tr>
<tr>
<td>IDISS</td>
<td>2</td>
<td>ARC3D dissipation</td>
</tr>
<tr>
<td>BIMIN</td>
<td>1.0</td>
<td>No low-Mach preconditioning</td>
</tr>
<tr>
<td>TFOSO</td>
<td>1.0</td>
<td>Euler implicit first-order time</td>
</tr>
<tr>
<td>ISPEC</td>
<td>2</td>
<td>Direction-specific spectral radius</td>
</tr>
<tr>
<td>SMOO</td>
<td>0.0</td>
<td>No smoothing modification</td>
</tr>
<tr>
<td>DIS2</td>
<td>2.0</td>
<td>User input coefficient, defaults to 2.0</td>
</tr>
<tr>
<td>DIS4</td>
<td>0.04</td>
<td>User input coefficient, defaults to 0.04</td>
</tr>
</tbody>
</table>

Figure 3.1: Outline of important equivalent OVERFLOW parameters when running IBSEN.

3.3 Oversetting

Overset (also known as “multiblock,” “Chimera,” or “composite overlapping”) grids provide a simple means of parallelization, as well as a way to control local grid density. Instead of using a stretched grid to place more points in a region of interest, an additional grid is created with the desired resolution, existing only in a small region of the field. Uniformly-spaced Cartesian meshes have the additional advantage of providing superior control and accuracy for acoustic wave propagation through the solution field, an important property when using permeable acoustic data surfaces that do not coincide with solid walls in the simulation. This “overset” mesh lies on top of whatever background meshes exist, and supersedes those meshes in the flow solution procedure. Data is communicated between the various meshes at their boundary cells using trilinear interpolation. To parallelize, any given mesh is completely independent of all others, and can exist on any processor. The Message Passing Interface (MPI) li-
library is used to facilitate communication between meshes when they lie on dif-
ferent processors. In this type of parallelism the speedup is achieved by placing
individual grids on separate processors and then sending “messages” between
adjacent grids. These messages contain the information necessary to compute
the solution at the overset boundaries.

3.4 Automatic Off-Body Grid Generation

To simplify the setup of large 3D cases, a relatively simple automatic off-body
grid generation system has been implemented for simplifying case inputs. The
generator automatically creates a sequence of grids that successively coarsen the
grid resolution and increase the domain size using minimal overlap between
regions. Figure 3.2 shows an example of the type of grid system generated by
this grid generator. Multiple independent generators can be used to fill complex
regions.

The grid generator supports differing the number of levels expanded in each
direction, the number of points in each level, and a number of other parameters
that give the user a relatively fine-grained control over the exact grid generated.
Overlap points are minimized and the spacing is adjusted to ensure exact over-
lap between the grids. By ensuring that as many mesh points as possible coin-
cide between the grids (the finer grid being a simple subdivision of the coarser
one in the overlap region) the amount of solution interpolation needed is min-
imized, improving both the speed and accuracy of the solution in the overlap
regions.
3.5 Load Balancing

One critical element for a solver designed to run very large cases is ensuring optimal use of the available computational resources. Modern computing clusters usually employ a combination of shared-memory and distributed-memory architectures, sometimes with additional processing units specialized for specific tasks (for example, Graphics Processing Unit (GPU) computing as demonstrated in Ref. 121). To take advantage of these resources, several types of load-balancing must be employed, at various levels.

The primary load-balancing in IBSEN is accomplished using grid splitting: the user-input grid system is modified at runtime to fill all of the available processors. For example, if the user inputs two grids, and four processors are available, the two grids are each split once, giving four grids. Taken one step further,
the code calculates the number of points in each grid and attempts to achieve some “optimal” grid splitting to ensure that all processors are filled. In reality, ensuring that all processors have exactly the same number of gridpoints to solve is not feasible: in the current implementation, a target of 80% efficiency is set. That is, the least-filled processor has at worst 80% of the number of gridpoints as the most filled processor. The grid splitting algorithm iterates until this criterion is met. The disadvantage of grid splitting is that in creating the new grids, overlap cells must be added to each to enable overset communication. In addition, this overlap increases the communication requirements: the amount of splitting that will give the optimal increase in overall run efficiency is not well-defined and depends on the exact architecture of the system.

In addition to the grid-splitting, more fine-grained parallelism can be achieved on shared-memory architectures through the use of OpenMP directives (Ref. 122). OpenMP differs from MPI in that it is designed to use a shared-memory architecture to perform parallel computations “in place.” For example, within the solution of a single grid there are steps that may be performed in parallel. One of these is that during the implicit solution procedure for a given direction \((\xi, \eta, \zeta)\) each row of the computational mesh is solved independently before being combine and cycling to the next direction. It is possible for the computation to be subdivided such that one processor computes the first row, another processor the second row, another the third, and so forth, each time directly placing the results in the required storage location. This is inefficient when using MPI due to the overhead required to pass the necessary information between processors rather; in OpenMP, since the results of the computation are simply stored directly in the appropriate location, this
overhead is not a factor. Experimental support for OpenMP has been enabled in the code, but further investigation of the optimal number of OpenMP threads versus MPI processes is required.

Finally, several sections of the code are amenable to GPU computing. In particular, in the implicit flow solver the linear algebra calculations are amenable to GPU-optimized libraries that use the processing power of the graphics card to increase the speed of the simulation. This is particularly valuable because GPUs are now frequently available on the computational clusters used to conduct the simulations. In addition, many of the cases considered in this dissertation are amenable to being run on modern desktop computing hardware, which typically includes a GPU. Although GPU computing is not yet implemented in the flow solver, it is expected that significant speedup is possible using this technique.

3.6 Support for CAD Input

Modern design begins by creating a geometry in a Computer Aided Design (CAD) package such as CATIA, ProE, AutoCAD, etc. In the CAD package each part of the geometry is analytically-defined, with hundreds or thousands of types of surface construction available. For example, various types of 3D spline surfaces may be used to represent the complex curves of a helicopter fuselage. Complex sets of equations are used internally to ensure proper alignment of the parts, and geometric tolerances are inherently included along with this information. Each CAD vendor uses a proprietary file format to store this complex data, so directly supporting all of those is infeasible, even for major commer-
cial CFD packages. Two alternatives exist since nearly all modern CAD packages are capable of exporting to both the IGES and STL file formats. The IGES (Initial Graphics Exchange Specification) standard is a very complex file format designed to support all analytical surface definition types from all CAD packages, and as such it is essentially a union of all the functionality from the most commonly-available CAD packages. Its successor, STEP (Standard for the Exchange of Product model data), has been implemented in some software, but is not as ubiquitous. Some software is available (both commercial and freely-available) for handling these analytical file formats, but the libraries are extremely complex and including them in IBSEN was infeasible for the present research. The STL (STereoLithography) file format takes a different approach, exporting the complex geometries tesselated into a set of triangles. There is an inevitable loss of fidelity when converting the analytical geometries into the discrete triangulated representation, but the resulting file format is extremely simple and very easy to input and manage. The major downside of using a triangulated data surface is the introduction of various types of degeneracy in the geometry. In general, triangulated data surfaces require a substantial amount of manual clean-up before they can be used in traditional body-fitted structured CFD packages. This step is generally combined with the generation of the structured surface grids and is typically a manual process that for complex geometries can take weeks. However, for an immersed boundary code some of these degeneracies can be overcome automatically by the geometry-handling code and others simply ignored. For this research a basic set of STL-handling routines has been implemented to substantially reduce the time required for a design engineer to set up the case.
3.6.1 Common Errors in Discretized Geometry

The most common error is for a triangle vertex that is supposed to be shared by adjacent triangles to be slightly offset so that the point is not, to floating-point accuracy, exactly the same. To handle this type of problem, some kind of point merging strategy must be employed. Because geometry handling is a relatively insignificant cost in the overall IBSEN runtime, a simple algorithm is chosen that looks at clusters of nearby points and merges them all to their average location if they all lie within a certain radius (internally defined as 10% of the length of the shortest non-zero line segment).

Other common errors include overlapping triangles, gaps between triangles, and curves that are discretized with differing resolutions on each side, resulting in a mismatch in the geometry. In this flow solver these conditions are not treated directly by the geometry-handling code, but are rather handled by attempting to minimize their impact on the flow solution. To do this, slower, brute-force algorithms are used instead of the more efficient (but more sensitive) optimal algorithms; numerous redundant computations are made in an attempt the minimize the number of degeneracies and their impact on the flow.

One common problem is when calculating whether a given point is interior or exterior to the geometry. If the ray cast from that point manages to slip through one of the minuscule cracks in a degenerate geometry, or passes through two slightly (and mistakenly) overlapping triangles, that point’s status will be incorrect. This does not, however, affect the status of any adjacent points, which are individually checked rather than relying on algorithms that assume that two adjacent points with no surface in between must have the same status (while this is intuitively obvious, it is also a fragile algorithm in the face of...
this type of degeneracy). In the final immersed boundary implementation presented below, in addition to detecting interior and exterior status of points, an additional check is performed to determine if a derivative calculation can use adjacent points. The edge joining each pair of points is individually checked for intersection with the geometry. If it intersects, then regardless of the interior/exterior status of either point, no derivative can be taken that involves both points. This prevents points near the surface, and erroneously detected as exterior, from influencing the flow solution, and represents one of the fundamental differences between the solution techniques used in this research as compared to previous methods.

Fig. 3.3 shows an example of an airfoil immersed in the grid. In this case, point A lies outside the body, and point B lies inside the body. A conventional immersed boundary simulation would choose to treat points A and B as boundary points on the basis of the interior/exterior status of the points alone. In the new technique detailed in Section 3.7.7, the solution procedure does not rely solely on the correct determination of point B’s interior status. During the initialization of the flow solver, every line segment connecting two adjacent gridpoints is checked for intersection with the geometry: for example, segment AB in the figure. If the segment is found to intersect, then the flow solution procedure behaves as though line segment AB does not exist, and points A and B are both treated as boundary points, their values set by the new immersed boundary method. In particular, the flow solution procedure will not allow the calculation of a value at point A using a finite difference stencil that involves point B, and vice versa. By adding this additional intersection calculation the overall robustness of the code is improved (the line segment connecting the gridpoints
Figure 3.3: Line segment AB is directly checked for intersection with the geometry surface, rather than relying on the interior/exterior calculation to succeed at point B.

is much shorter than the ray projected during the interior/exterior calculation, so is less susceptible to floating point imprecision errors), as well as having the added benefit of naturally handling cases such as airfoil trailing edges, where it is possible that neither of the points in question lie inside the body, but the line segment connecting them intersects the thin region nevertheless. This mechanism effectively results in a method that does not actually require interior/exterior calculations at all. In the final chosen immersed boundary method detailed below the interior/exterior calculations are performed to improve flow visualizations, and to allow the elimination of definitively interior points from the solution for efficiency and stability, but are not directly related to the immersed boundary condition application.

Finally, due to the smoothing nature of the surface boundary condition application used in this research (and detailed in Section 3.7.7, below), any degen-
eracies that remain tend to only influence the very local flow, and in general have a small impact on the overall flow solution. This of course depends on the exact degeneracy and the nature of the nearby flowfield (high-gradient regions are more sensitive to errors than low-gradient regions, for example), but the final code has been designed to be as tolerant of these degeneracies as possible. This is principally accomplished by requiring independent calculation of the interior/exterior status of every point regardless of its neighbors’ statuses, as well as the independent calculation of whether the line segment connecting each pair of gridpoints intersects a solid surface. Coupled with the highly-robust immersed boundary technique shown in Section 3.7.7, the possible effect of any errors in the aforementioned calculations are limited to the local region immediately surrounding the degeneracy.

3.6.2 Efficient Triangulated Surface Storage and Access

The first step to handling a triangulated data surface is to determine how to store it. There are many data structures available for this task, employing various schemes for optimizing various algorithms. Some schemes are designed for a minimal memory footprint, other for fast determination of neighboring triangle information, others for fast determination of the triangles in a given region of space. For this research, where the memory associated with the geometry is much lower than the memory associated with the flow solution, the amount of memory used is not a significant factor. In addition, quickly determining adjacent triangles is useful for detecting sharp corners in the geometry (discussed in Section 3.6.5), while quickly determining the triangles in a volume of space is important for determining whether a grid point lies inside or outside the trian-
gulated surface (discussed in Section 3.6.3).

An alternating tree is similar in principle to a binary search tree executed in multiple dimensions (Ref. 123). It stores multi-dimensional objects in such a way that they can be efficiently retrieved based on their location in space. In this code the ADT is used to narrow the number of triangles that must be tested for intersection with a given ray, as that intersection process is quite expensive. It reduces the asymptotic complexity of the interior/exterior detection problem from $O(n)$ to $O(\log n)$ by eliminating the vast majority of the triangles which cannot intersect the test ray. Note that it does not reduce the problem to those triangles whose bounding box intersects the ray, but rather depending on the construction order of the tree, to those triangles whose bounding box may intersect the tree. It discards those whose position in the tree indicate that they absolutely cannot intersect, leaving the remaining triangles to be rigorously tested. This set includes all triangles whose bounding box intersects the ray, as well as several others that are encountered while traversing the tree. A simple bounding box test can exclude those extra triangles just prior to testing against the ray, if desired.

The tree is constructed by looping over a list of triangles and inserting them one by one based on their bounding box. To begin, the triangles’ coordinates are normalized to lie in the unit cube (this simplifies extraction from the tree during intersection testing). The first triangle is simply inserted as the head node of the tree. The next triangle’s $x_{\text{min}}$ coordinate is examined, and if it is less than 0.5 the triangle is inserted at the left branch of the tree. If it is greater than or equal to 0.5 it is inserted at the right branch. For the next triangle, its $x_{\text{min}}$ coordinate is examined, and if it is less than 0.5 the tree is traversed down a level, to the
previously-inserted node. The current triangle’s $y_{\text{min}}$ coordinate is examined at this level and in the same manner it is inserted at either the left or right branch of the tree. This process continues, always traversing down the tree until an open node is found. At the third level of the tree the $z_{\text{min}}$ coordinate is examined, at the fourth level the $x_{\text{max}}$, at the fifth level the $y_{\text{max}}$, and finally at the sixth level the $z_{\text{max}}$. At the seventh level the tests wrap around and the $x_{\text{min}}$ is tested again. The tree is constructed following this pattern until all triangles are placed in the tree. For more detailed explanation of the technique please see Ref. 123.

3.6.3 Interior-Exterior Testing

As mentioned above, although the interior/exterior detection is not critical to the flow solution procedure in the selected immersed boundary method, the calculation is performed as part of the solution procedure nevertheless, allowing the exclusion of interior cells from the solution procedure for improved efficiency and stability. In addition, most of the other immersed boundary techniques require the interior/exterior detection to be accurate, so for comparison purposes it was important to implement as robust an algorithm as possible. In the interest of computational efficiency it is important to choose an algorithm that scales well with grid and geometry size and runs in a reasonable amount of time. While for the cases presented in this research the immersed boundaries are stationary relative to their grids, the solver itself supports immersed boundaries that are moving relative to the grids. Interior-point detection is performed using a ray-casting technique. Once the ADT is created (one time, on program initialization), the ADT scheme allows searches for potential intersecting triangles in $O(\log n)$ time. The intersection between a ray cast from the grid
node being tested and the triangles representing the geometry is then calculated, using the ADT to determine possible intersections before the arithmetic is performed. Shewchuk’s binary predicates for ray-triangle intersections are then be used to compute the number of intersections (Ref. 124). If this number is odd, the point lies within the geometry, if the number is even, the point is outside, as shown in Fig. 3.4. If the point lies exactly on the surface the result is undefined (the point is technically neither interior nor exterior), and for the purposes of this research the point is treated as exterior. Immersed boundary algorithms must be designed in such a way that they cope with this scenario. Note that this method implicitly assumes that the geometry is a “wetted surface”—that is, no part of a surface lies inside another surface. The non-wetted-surface case can be handled (see Ref. 36, for example), but since the interior/exterior detection is not critical for the operation of the flow solver more advanced techniques are not implemented in this research. The overall complexity of the algorithm is $O(n \log m)$ where $n$ is the number of grid cells and $m$ the number of triangles in the geometry.

3.6.4 Robust Intersection Testing

Although interior/exterior detection is not critical to the new immersed boundary technique presented here, the location of the boundary must be well-identified, and in particular any grid cell edge that intersects the geometry must be flagged. The same code that performs the ray-triangle intersection for the interior/exterior code is used to perform the line segment-triangle intersection used in the new technique. While some degeneracies in the flagging are tolerated by the new solution mechanism, the intersections should be identified
Figure 3.4: An example of inside/outside detection using ray-casting. The upper ray intersects the geometry an odd number of times (once) indicating that the originating point is interior, while the lower ray intersects the geometry an even number of times (twice), indicating that the originating point is exterior.

correctly whenever possible (in particular when comparing the new technique to previous techniques which were much less tolerant of degeneracies). There are numerous line segment-triangle intersection tests available with varying degrees of accuracy and optimization (see, for example, Refs. 36,125,126). Because a failure to detect an intersection could potentially lead to an unstable flow solution (in the new technique) or no solution at all (in previous techniques) it is advantageous to use as robust an algorithm as possible. This led to the selection of the method used by Aftosmis et al. in Ref. 36. This technique calculates the signed area of three tetrahedra formed by the three points of the triangle and the two endpoints of the ray being tested. If the area of each has the same sign, then the ray intersects the triangle. In Ref. 36 this method’s robustness is supplied by Shewchuk’s adaptive-precision Orient3D predicate (Ref. 124).
Unfortunately this predicate requires strictly IEEE-compliant 32-bit arithmetic, limiting its usefulness on modern processors and with modern compilers (IEEE 754 requires that single-precision floating point arithmetic use exactly 32 bits, but most modern processors use more precision internally to reduce round-off error—turning this feature off results in dramatically slower code). For the present implementation a more straightforward (though less efficient) approach is taken.

The volume of a tetrahedron is directly proportional to the determinant of the matrix formed by its vertices:

\[
V = \frac{1}{6} \begin{vmatrix}
    a_x & a_y & a_z & 1 \\
    b_x & b_y & b_z & 1 \\
    c_x & c_y & c_z & 1 \\
    d_x & d_y & d_z & 1
\end{vmatrix}
\] (3.28)

where \(a, b, c\) and \(d\) are the vertices of the tetrahedra. Shewchuk’s test determines the sign of the determinant using an adaptive precision algorithm that uses only as many digits as a forward-looking error analysis demonstrates are needed. While slower than using standard single- or double-precision arithmetic, it provides a measure of security when using other algorithms that rely on its correctness. The test takes advantage of the fact that the exact result of the determinant is not required: only the correct sign. Determinants not near zero, therefore, are not calculated with the high-precision routines. For determinants near zero, an error analysis determines if the result can be trusted, and if not, continually increases the precision until it can be. In the present application, rather than using the sophisticated adaptive precision arithmetic, if
the result of the standard non-IEEE-compliant 64-bit determinant calculation is found to be close enough to zero that it falls within the round-off error bounds of that particular processor, the determinant is recalculated with 128-bit floating point numbers that use exact rounding [via the GNU Multiple Precision (GMP) (Ref. 127) and Multiple-Precision Floating-point with correct Rounding (MPFR) (Ref. 128) libraries]. These libraries ensure that if the determinant is smaller than the roundoff error of the calculation the result will be exactly zero.

In the case where the high-precision determinant is found to be zero, a naive interior/exterior-detection algorithm will fail. This case occurs when the tested line segment intersects the edge between two triangles (rather than the normal case where it intersects somewhere on the face of a single triangle). For the ray-casting algorithm to work properly, if a ray intersects precisely with an edge shared by two triangles, the algorithm should only report a single intersection. This requirement can be met using an algorithm developed by Mücke and Edelsbrunner called Simulation of Simplicity (SoS) (Refs. 129, 130). This technique develops a purely symbolic perturbation of the geometry that is mathematically guaranteed to result in consistent intersection results. In particular, if a ray intersects the line between two triangles, or the endpoint joining several triangles, it is guaranteed to only return a single point of intersection, without requiring any individual special cases.

In the case of mistakenly overlapping triangles the intersection code will report two intersections, causing the interior/exterior detection code to fail. This degeneracy is naturally handled by the use of the grid cell edge intersection test described in Section 3.6.1 during the solution procedure, however: regardless of the number of intersections the flow solver will prevent the derivative being
taken using the two endpoints of that edge and will instead require that they both be set via the new immersed boundary technique. This natural tolerance of certain types of degeneracies is one advantage the new technique has over the others presented.

3.6.5 Surface Curvature Calculation

For some surface boundary treatments the local surface curvature is required. For a surface in 3D space the curvature has two components: for this research the minimum and maximum principle curvatures \((k_1, k_2)\) and principle curvature directions \((\hat{e}_1, \hat{e}_2)\) are used. In an analytically-defined surface these curvatures are calculated as the Eigenvalues of the Second Fundamental Form (see e.g. Ref. 131 for details). For a discrete surface, however, the surface curvature is not well-defined. Each triangular facet is a plane of zero curvature, joined to another facet at an edge with infinite curvature. What is required for the CFD calculation is the surface curvature of the original surface, not the discretized surface. To approximate that curvature, the following algorithm is used:

1. Identify sharp edges in the geometry. For this research a sharp edge is defined as an edge where two triangles intersect at an angle greater than \(45^\circ\).

2. At each surface node, identify at least six neighboring points that do not lie on the opposite side of a sharp edge.

3. Construct an analytical surface centered at the node in question using a 2D quadratic fit to the neighboring points.
4. Calculate the principal curvature analytically from the quadratic surface.

The last two steps are implemented using the Computational Geometry Algorithms Library (CGAL), though some pre- and post-processing of the data is required to obtain usable results. Before calling the CGAL library the points are sorted by proximity to the point at which the curvature determination is needed, and the results are post-processed to ensure that the returned normal vector points outward from the geometry (if the vector does not point outside when returned from CGAL, the returned coordinate system is rotated such that it does). In addition, some versions of CGAL return the normal vector and principle curvatures in arbitrary order, so this is checked and corrected as well. Finally, very small curvatures are manually forced to zero to prevent floating point errors accumulating when solving the flow over flat surfaces.

3.6.6 Closest Surface Point Identification

During the flow solution procedure the surface treatment techniques typically require information about the surface point that lies closest to the grid point in question. For a discretized surface, this is a complex calculation. The first step is to identify the subset of triangles which may contain the point closest to the specified grid point using the ADT. From this list of triangles, the closest point of approach to the grid point is calculated: in general this point may lie on the face of the triangle, or on one of the edges or nodes. Once each triangle has been checked, the one with the shortest distance is selected.

Once the triangle and closest point have been identified, the local surface curvature must be determined. The curvature was calculated at each triangle
node as a pre-processing step when the geometry was read. There are three basic cases (listed here from most common to least common):

**Closest point is on the triangle face.** The simplest and most common case, if the closest point lies on the face of the triangle it cannot lie on a sharp edge (since it does not lie on an edge at all) and therefore the curvature is finite (or zero), even if one of the triangle’s edges is a sharp corner. To calculate the curvature at this location, the curvatures from the nodes are simply interpolated in barycentric coordinates to the point on the triangle’s face.

**Closest point is on a triangle edge.** If the closest point does lie on a triangle edge, it is possible that the curvature is infinite, if the edge is sharp. When calculating the curvature at this point the edge structure is checked to see if it was previously detected as a sharp edge. If so, infinity is returned for the maximum principal curvature, otherwise the values from the two adjacent nodes are used to interpolate to the closest point location and that value is returned.

**Closest point is on a triangle node.** If the closest point is a triangle node it may also be on a sharp edge. All edges the node is a member of are checked and if any of them is sharp, infinity is returned for the maximum principle curvature. If not, the curvature calculated for that node is returned.

Finally, it should be noted that in addition to the principle curvatures, the local normal vector and principle curvature directions are calculated and output using approximately the same logic. Additional processing is required to ensure that the directions are consistent, however, since a principle curvature
vector may point in either the positive or negative direction and this case must be resolved correctly when averaging to ensure the correct result is obtained.

3.7 Treatment of Immersed Boundaries

In the immersed boundary method, the body boundary conditions are generally not applied directly at the body surface (which does not necessarily correspond exactly to the grid), but must be allowed to influence the flow solution through their application to cells near the surface of the body. Numerous researchers have developed techniques for representing the surface. These range from finite-volume, cut-cell “Cartesian Grid” methods to finite difference approaches that interpolate and/or extrapolate the body surface conditions into the flowfield. The action of the surface can be diffused over a band of several cells (e.g. Ref. 38), or concentrated at only the cells immediately adjacent to the surface (e.g. Ref. 48). In the present implementation the body boundary conditions are applied to the cells immediately adjacent to the surface location. Numerous techniques are examined in the following sections for their efficiency and accuracy when used on coarse grids for inviscid, compressible flow solutions. One of the aims of this research was to identify a surface treatment technique that extends well to coarse 3D grids over highly-complex geometries. Nearly all research to date on immersed boundaries focuses on grids that, if extrapolated to a rotor case, would result in hundreds of millions or perhaps billions of grid cells: this is clearly infeasible if the flow solver is to be used for design purposes. Therefore, an alternative approach was taken in this research. First, a flow solver framework capable of simulating the mo-
tions of arbitrarily complex geometries in motion was developed. The solver can provide time-dependent details of surface-line intersections, identify points on the body surface that are closest to a point in the flowfield, and provide complete surface curvature and motion information to the flow solution routines. Next, a set of geometries and grids for several complex rotorcraft cases were developed, using uniform, overset, moving Cartesian grids and targeting a realistic, computationally-feasible final case size (on the order of 10 million gridpoints). From these grids, which represent the actual target grid resolution that we would like a solution scheme to support, the existing literature was examined and numerous candidate techniques were identified. These techniques were then implemented in the flow solver and their behavior at the required grid resolutions was examined. Only techniques that are shown in the literature to be stable for rigid bodies were selected, and the focus was on the more recently-developed discrete-forcing penalization techniques (as opposed to those which distribute the force over a small band of cells near the body surface).

The methods are broken down into two categories based upon the fundamental basis of the method: penalization or reconstruction. In a penalization method, an additional forcing term is added to the momentum equations to simulate the force acting on the fluid due to the presence of the solid body. In a reconstruction method, the flowfield is reconstructed at the body surface and traditional CFD boundary conditions are applied; the surface data is then interpolated out to the surrounding grid cells.
3.7.1 General Advantages and Disadvantages of Penalization Methods

Penalization techniques are the “classical” immersed boundary method, designed for simplicity and ease of implementation. Originally designed for incompressible, viscous flows, in recent years researchers have extended them to compressible flows, though they are still quite focused on viscous solutions to the Navier-Stokes equations. Some limited research has been presented using the techniques in inviscid flows, though a no-slip boundary condition is still generally applied. See Chapter 2 for a review of the pertinent literature. The most basic feature of a penalization method is that the differentiation scheme proceeds through the grid as though no body is present, taking standard finite-difference derivatives even between points that lie within the body itself, or across a body surface. After the derivatives are calculated, the result is used to calculate a “penalization force” to apply to points lying within the body, forcing their momentum to a predetermined value. The aim of this forcing is to simulate the correct surface boundary conditions when the next derivative is taken, at the next timestep.

Mohd-Yusof’s formulation provides a useful starting point in understanding penalization methods. The immersed boundary method relies on a forcing function to represent the body in the momentum equation (Ref. 132):

\[ \frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_i} (\rho u_i) = 0 \] (3.29)

\[ \frac{\partial}{\partial t} (\rho u_i) + \frac{\partial}{\partial x_j} (\rho u_i u_j + p \delta_{ij}) = f_i \] (3.30)
where \( f_i \) is the forcing function used to model the body. Using the method suggested by Mohd-Yusof, the discretized momentum equation is

\[
\frac{\rho u_i^{n+1} - \rho u_i^n}{\Delta t} = \text{RHS}_i + f_i
\]  

(3.31)

where \( \text{RHS}_i \) contains the convective and pressure gradient terms and the body force, \( f_i \), is

\[
f_i = \begin{cases} 
0 & \text{for points outside the body,} \\
-\text{RHS}_i + \rho \left( v_{bi}^{n+1} - u_i^n \right) / \Delta t & \text{for points inside the body}
\end{cases}
\]

(3.32)

where \( v_{bi} \) is the velocity of the body. The effect of this term is to force the fluid inside the body to move with the speed of the body. In this basic formulation, no attempt is made to include information about the actual location of the surface, only that some points lie within the surface and some lie outside it. Various researchers have developed techniques for attempting to include more information about the surface location in the forcing function: these will be detailed below.

Penalization methods such as this have a disadvantage that is exacerbated by coarse grids: any part of a geometry that the fluid must flow around must have gridpoints inside it, and the number of points required increases as the order of the spatial derivatives increases. This presents a problem at airfoil trailing edges, thin geometries, or simply in regions where the grid is not sufficient to resolve the body geometry (perhaps because the user is not interested in the flow in those regions). Figure 3.5 shows a schematic of several types of problem areas.
(a) Thin body with single layer of gridpoints inside. A differentiation stencil selects points on the outside of the body on the other side of a solid wall that should not influence the flow.

(b) Thin body with no gridpoints inside. The penalization method cannot represent this geometry at all.

Figure 3.5: Degenerate cases in penalization methods.
In Fig. 3.5a a third-order differentiation stencil is formed that includes one
point inside a thin body, plus points on both sides of the body. Physically, no dif-
ferentiation should occur across body boundaries, but in a penalization method,
body boundaries are effectively ignored and stencils formed without regard to
the surface position. This degeneracy commonly occurs in a derivative across a
thin airfoil trailing edge, making it difficult to correctly represent the flowfield
in that region. In addition, discrepancies at the trailing edge can in some cases
strongly influence the remained of the flowfield. Anytime the differentiation
stencil is large enough to reach across a solid boundary and include flow points
outside that boundary, non-physical derivatives will be calculated.

In Fig. 3.5b, a thin body cuts through the grid in such a way that no grid-
points lie inside the body. A penalization method is completely unable to rep-
resent this type of geometry. This degeneracy can also commonly occur in thin
trailing edges, again resulting in poor flowfield representation.

Numerous researchers have attempted to resolve these problem areas by, for
example, introducing double-valued points at the body interior locations, etc.
Unfortunately, the introduction of these special cases at the body surface elimi-
nates much of the advantage that penalization techniques have over reconstruc-
tion methods (namely, their simplicity and elegance of operation). Once double-
valued points are introduced, the finite-difference scheme must be modified to
account for the presence of the body, introducing a branch in the code:

```c
if (StencilIntersectsBody()) {
    RHS = CalculateNearbodyFlux();
} else {
    RHS = CalculateRegularFlux();
}
```
Once this branch is introduced, there is little reason to continue with the present approximate treatment of the body boundary conditions. It could instead be replaced with:

```cpp
if (StencilIntersectsBody()) {
    RHS = 0
    Q = SetBoundaryCondition();
} else {
    RHS = CalculateRegularFlux();
}
```

In this case, more traditional CFD boundary conditions can be adapted to the Cartesian grid method: this is the basis of the reconstruction methods discussed below.

For the cases run in the following sections, two grids were used: “coarse” and “fine.” It should be noted, however, that because this research is focused on coarse grids, neither grid is fine enough to perform viscous flow calculations (in many cases seen below, even the fine grid is far coarser than the researchers who developed the technique intended to be used), and both grids are finer than the desired resolution for modeling a rotorcraft fuselage using an existing rotor-only grid system (the ultimate goal of this research). Figure 3.6 shows a close-up view of the inner grid for each system, with the body interior points blanked out. The coarse grid has approximately 22 grid cells across the airfoil thickness, and 125 cells chordwise; the fine grid is twice that resolution. Even coarser grids were attempted in many cases, but results were only obtained for the last method presented (the “inverse distance-weighted average” reconstruction method). All cases were run with a third-order upwind-biased differentiation scheme, Lax–Friedrichs flux splitting, and a second-order com-
Figure 3.6: Innermost grid configuration for boundary condition test cases.

pact Runge–Kutta timestepping scheme. The freestream Mach number was 0.2 and all cases were run non-dimensional. Cases that diverged are noted in the text.

3.7.2 Penalization Method 1: Discrete Forcing

The discrete forcing method developed by Mohd-Yusof (Ref. 48) calculates the force required to bring the fluid entering the body to rest and applies that force to the right-hand side of the Navier-Stokes equations. This is based on the porous media penalization technique first developed by Brinkman (Ref. 46) and effectively attempts to treat the body region as a zero-porosity region, but still modeled by the Navier-Stokes equations (with the inclusion of a body-force term). This technique is typically applied only to viscous flow solutions and
is usually coupled with a turbulence model to more correctly handle the exact location of the body surface. This is due to the extremely low-order surface treatment this method otherwise results in. In particular, Fig. 3.7 shows a high-order differentiation stencil at the body edge, incorporating two points in the field and two points in the body. Note that in applying this differentiation stencil, no attempt is made to enforce the correct derivative calculation at the actual location of the body surface, instead treating the gridpoints there as if they were a fluid and allowing them to influence the flow solution. If the body is stationary and a fine grid is used this results in a relatively small error as the velocity at the body surface will be nearly zero. With large grid cells, however, a substantial non-zero velocity may exist at the actual physical location of the body surface. With moving bodies the problem is compounded further by allowing the solid body particles within the geometry to influence the flow solution results in a physically inconsistent flux calculation (the influence of a solid body should only act at the interface between the body and the fluid).

For comparison purposes this technique has been implemented in the present inviscid flow solver: the effective Reynold’s number is grid-resolution-dependent due to the use of the no-slip boundary condition and the lack of viscous stress terms in the inviscid equations of motion.

Figures 3.8b and 3.8b show the coarse and fine grid lift convergence plots: it is clear that the lift for these cases never converges to a steady value, but instead oscillates around a mean, suggesting flow separation has occurred for both cases, though the fine grid cases results in smaller oscillations about a mean that is closer to the actual predicted lift value of \( C_l = 0.33 \) (from thin airfoil theory). Figures 3.9a and 3.9b show the coarse and fine grid \( C_P \) distributions
on the airfoil surface at the last timestep of each run along with a plot of an OVERFLOW calculation for the same case (run 2D, inviscid, on a coarse grid). Again, while the fine grid results are superior to those of the coarse grid, neither result is satisfactory, showing flow separation features at the trailing edge, and an overprediction of the lower surface peak pressure. Finally, Figs. 3.10a and 3.10b show the pressure distribution in the field, clearly demonstrating that the flow has fully separated on the upper surface of the airfoil. The separation characteristics depend on the grid spacing, with the fine grid showing smaller, higher-frequency shed vortices.

In general it is seen that this technique is inadequate for inviscid flow solutions, at least on coarse grids. While when grid resolution is increased the so-
olution improves somewhat, the projected grid density required to achieve even marginal solutions is prohibitive for full-configuration simulations of rotorcraft using an explicit flow solver with presently-available computing resources. This technique is clearly targeted, and most appropriate for, viscous flow solutions, which also necessitate much finer grid spacing. The use of a no-slip wall boundary condition in an inviscid solver of this nature leads to unrealistic flow solutions, separating even in this thin-airfoil result at 3 degrees angle of attack.
Figure 3.8: Lift coefficient vs. time for discrete-forcing penalization cases. Dashed line shows nominal thin-arifoil result, $C_l = 0.33$.
Figure 3.9: Pressure coefficient distribution for the discrete-forcing penalization case. Solid line is the inviscid OVERFLOW solution, symbols are the IBSEN solution.
Figure 3.10: Pressure field for discrete-forcing penalization case.
3.7.3 Penalization Method 2: Volume Fraction

The volume fraction technique (Ref. 51) attempts to reduce some of the dependency on a turbulence model at the surface for modeling when using the discrete forcing approach. Instead of calculating a force necessary to bring the entire volume of fluid represented by a given interior grid cell to rest, an approximation of the volume that actually lies interior to the body is calculated and the force necessary to bring only this fluid mass to rest is applied. Figure 3.11 shows a schematic of this approach.

This results in a velocity field that at least partially accounts for the actual physical location of the surface directly in the flow simulation, even without a turbulence wall function, etc.

Figures 3.12, 3.13, and 3.14 show the results for the volume fraction case. This technique yields results that are almost exactly the same as the more straightforward discrete forcing approach. Like the discrete forcing approach, the major stumbling block for this technique is its inability to model the proper slip wall non-penetration condition when used with an inviscid flow solver. The “high-Reynolds number approximation” of applying a no-slip wall boundary condition is simply inadequate when coupled with a coarse grid solution. The flow separates immediately around the shoulders of even this very thin airfoil at three degrees angle of attack.
Figure 3.11: The volume fraction technique modifies the force applied to more accurately reflect the actual physical location of the body surface.
Figure 3.12: Lift coefficient vs. time for volume fraction penalization cases. Dashed line shows nominal thin-arifoil result, $C_l = 0.33$
Figure 3.13: Pressure coefficient distribution for the volume fraction penalization case. Solid line is the inviscid OVERFLOW solution, symbols are the IBSEN solution.
Figure 3.14: Pressure field for volume fraction penalization case.
3.7.4 Penalization Method 3: Velocity Mirroring

In the velocity mirroring technique originally demonstrated by Fadlun and others (Refs. 29, 51, 133), rather than applying a force to bring the fluid to rest inside the body, a force is applied to set the velocity opposite to that across the body’s surface. Figure 3.15 shows a schematic of this technique.

The general principle is that differentiation across the body surface during the timestepping should simulate the surface non-penetration condition. In the mirroring method, the known velocities in the field are reflected across the body boundary and then negated (see the vectors mark $v = -v(1)$ and $v = -v(2)$ in the figure). The velocity from these points is then used with the field points to interpolate to the gridpoints inside the body. If the interpolation matches with the differentiation stencil, a zero-velocity condition is enforced at the body surface. Fictitious flow inside the body is then ignored. One advantage of this technique is that a true slip-wall non-penetration boundary condition can be applied if desired, rather than forcing the velocity to zero. In that case, only the normal velocity component is negated, the other components are simply mirrored.

In general it is very challenging to mirror a 3D flowfield across the body for more than a single point and achieve a physically consistent flowfield. In addition, attempts at using higher-order methods at the body surface using this technique generally diverged. Demonstration results have only been achieved by reducing the body surface treatment to only a single layer of cells.

Coarse and fine grid results are shown in Figs. 3.16, 3.17, and 3.18 for a single mirrored layer using a no-slip surface condition. In general this technique was found to be highly unstable for coarse grids, and only marginally stable for
finer grids: cases were run with multiple layers and with slip-wall boundaries, but without special treatment at the airfoil trailing edge the solution diverges quite quickly. Again with the no-slip boundary condition the solution does not converge to a steady-state result, and in fact the results are considerably worse than with the simpler techniques.

There are several possible explanations for this. First, the mirroring across the surface is degenerate in the case of a thin trailing edge such as this: special treatment is required to overcome this problem. Those using this technique typically resort to using double- (and triple- and quadruple- etc.) valued points and other such special cases in troublesome regions such as thin areas of the geometry. This in effect eliminates the penalization method’s advantage of being
able to use an unaltered differentiation scheme across the body boundary, turning the method into a reconstruction method that simply uses the body interior points as convenient storage locations. Further investigation may be warranted, but at this time the additional complexity necessary to handle the “ghost-cell method” of Dadone and Grossman (Ref. 13) is unimplemented. In particular, the “ghost-cell method” relies on multiple-valued points at sharp corners and other regions of high convexity.

Second, a third-order upwind-biased spatial differentiation scheme was used, requiring a four-point stencil for each evaluation. At the body surface, to calculate the derivative at the exterior of the surface, two points inside the surface are used, but the mirroring technique was only stable for a single layer of cells (diverging quickly at the trailing edge when multiple layers were requested). Therefore, the flow solution inside the body is allowed to influence the flowfield exterior to it, resulting in non-physical behavior. Again, adding the capability for multi-valued points may alleviate some of this problem. When run for a flow over a sphere (which does not require multiple-valued points) the method is stable and converges to a reasonable steady state solution (as do all of the penalization methods).
Figure 3.16: Lift coefficient vs. time for mirroring penalization cases. Dashed line shows nominal thin-arifoil result, $C_l = 0.33$
Figure 3.17: Pressure coefficient distribution for the mirroring penalization case. Solid line is the inviscid OVERFLOW solution, symbols are the IBSEN solution.
Figure 3.18: Pressure field for mirroring penalization case.
3.7.5 General Advantages and Disadvantages of Reconstruction Methods

Reconstruction methods are generally substantially more complex to implement than penalization methods due to the need for solution information at the body surface. While it is common in a penalization method to implement some technique for deriving surface pressures, they are not used to evolve the flow solution, so the results are not sensitive to the technique chosen. In a reconstruction method, the surface pressure extraction is an integral part of the flow solution procedure, and the choice of technique can strongly influence the resulting flow solution and its stability. In addition, special treatment of the near-surface grid-points is always required in the flux calculations when using a reconstruction technique. However, a reconstruction method eliminates any “stairstepping” effects that may arise in some types of penalization methods, and is in theory capable of correctly enforcing the surface non-penetration condition. Finally, slip-wall boundary conditions are possible using this technique, enabling more physically-consistent solutions to inviscid flow problems.

3.7.6 Reconstruction Method 1: Surface Normal Extrapolation

The first reconstruction technique examined is that of Gilmanov, et al. (Refs. 27,53). In this technique a linear interpolation/extrapolation procedure is used to enforce a linear velocity profile at the physical surface of the geometry by setting the velocity at the surrounding points such that the velocity would be zero (in the no-slip, high-Reynolds number approximation case) or parallel to the surface (in the fully inviscid, slip-wall case) at the location of the actual
surface. Figure 3.19 shows a diagram of the linear interpolation/extrapolation technique. In it, each gridpoint that is exterior to the body but adjacent to it is marked as an “edge” point where the boundary conditions will be applied (the red point in the figure). At each gridpoint, the nearest body surface position is calculated (blue point in the figure). Along smooth body walls the vector from this point to the grid point in question is a normal vector to the surface. Projecting this normal vector out into the field allows the interpolation of field data onto points along the normal, so we choose the closest two independent points (the green points in the figure), which in general are not themselves gridpoints, but lie on the faces of cells where the data is known from the flow simulation (the yellow points mark where data is known and can be interpolated into the green points). The data from the field is first interpolated onto the normal vector at those locations. Then, depending of the flow quantity in question (density, momentum, or energy), and appropriate algorithm is chosen to set the value at the boundary point.
For an inviscid, slip-wall boundary condition, the only known quantity at the wall is the normal velocity, \( \mathbf{u} \cdot \mathbf{n} = \mathbf{v} \cdot \mathbf{n} \), where \( \mathbf{u} \) is the local flow velocity, \( \mathbf{n} \) is a local normal vector to the surface, and \( \mathbf{v} \) is the local surface velocity. All other parameters must be extrapolated from the flow field itself. To accomplish this, an extrapolation from the known points along the normal vector is used for the complete velocity vector. The normal momentum is then subtracted from the resulting term. An interpolation is then performed between the known field values along the normal vector and the reconstructed surface value to yield a boundary condition value for the gridpoint in question. The density is extrapolated to the gridpoint from the known points on the normal vector, and the pressure at the surface is either simply extrapolated, or calculated by solving the normal momentum equation at the surface point using the extrapolated values for density and velocity.

At the surface of a moving body the normal momentum equation can be written as an equation for the surface pressure normal derivative,

\[
\frac{\partial p}{\partial n} = \rho \left[ -\frac{\partial u_n}{\partial t} + k_1 u_1^2 + k_2 u_2^2 \right]
\]  

(3.33)

where the \( k_1 \) and \( k_2 \) terms are the minimum and maximum principal curvatures of the body at this location, and \( u_1 \) and \( u_2 \) are the velocities in the minimum and maximum directions of principal curvature.

This results in an immersed boundary condition application as follows:

1. **Identify “edge points” in grid.** The list of gridpoints that lies just outside the body is created. These are the points the boundary conditions will actually be applied at, referred to here as “edge points.” These are the
points at which we actually set the boundary values and which influence
the flow solution. They are identified using a brute-force search for grid
points that are connected to an adjacent grid point by a line segment that
intersects the body. Only points exterior to the body surface are treated
as edge points: any corresponding interior edge points are ignored for the
purpose of the simulation.

2. **Find the nearest surface point to each edge point.** In order to avoid any
sort of stairstepping effects, the next step is to determine exactly how far
from the surface each edge point lies. At the closest point of approach
on a body, we need the coordinate system formed by the normal vector
and directions of principal curvature, as well as the principle curvatures
themselves (for use in the pressure boundary condition derived above).
The first step is identifying the point of closest approach. For analytically-
defined geometries this is usually very simple, whereas for discrete ge-
ometries it is a difficult and expensive task. Note that the “closest point”
does not mean the closest point that is used to define the discretized surface. The
actual closest point will in general lie somewhere in the middle of a sur-
face facet, and will most times not coincide with a point used to define
the geometry. This removes any strong dependence of the flowfield on
the number of points used to define the body, as long as regions of high
curvature are adequately resolved. In addition to simply calculating the
closest point of approach, calculating the curvature on a discrete surface
is a challenging problem, so in the present implementation an external
library is relied upon for this functionality. The library used, the “Com-
putational Geometry Algorithms Library (CGAL),” is a very large, com-
plicated library, so in the future it would be advantageous to implement the necessary algorithm directly in IBSEN, rather than relying on a library. Note that in the present implementations, no wetted-surface extraction is performed, so if two bodies intersect in space, the coordinate system and curvature values returned by these functions along that intersection line will simply be the values from one of the bodies, not the sharp corner that would be expected at the intersection.

3. **Extrapolate flow solution to surface point.** Once the surface point has been identified, the code extrapolates from the flowfield onto the body surface. The body boundary conditions are then applied at this surface point using the technique of extrapolating along the normal vector, detailed above.

4. **Apply boundary condition on edge points.** The final step is to set the appropriate flow solution values at the edge points. This is a simple linear interpolation between the nearest surface point (where in the previous step we extrapolated the flow and applied the body boundary conditions) and the nearest flow solution point along the body normal vector. Note that this interpolation also uses the surface normal extrapolation technique for consistency.

### 3.7.6.1 Correction for Unresolved Body Features

The surface normal extrapolation reconstruction technique relies on the ability to identify two independent points along a normal to the body where the flowfield is known from the main solution procedure. There are two circumstances
under which this technique will fail: first, if the body contains small features that are unresolved by the grid, it is possible for a grid point to lie in a region where the normal vector from the body surface intersects the body again before two independent flow point can be identified. Figure 3.20 shows the most basic case of a small divot in the body where the reconstruction may fail. The shaded region shows the interior of the body. The blue points in that region are interior points. Points that are adjacent to the blue points but are not interior themselves, colored red here, are classified as edge points and must be reconstructed. The remaining points, colored green, are normal flow cells. During the main surface normal extrapolation reconstruction technique, the highlighted point A is recognized as a body edge node, where we must reconstruct the boundary condition.
Figure 3.21: With the non-resolved points filled in, the body appears to the flow solution procedure to be smooth over the top of the unresolved cavity.

Projecting a normal vector from the surface, through point $A$, the code will identify points $B$–$F$ as the closest face points to point $A$. Point $B$ can be interpolated to because the two required points are classified as normal flow nodes, but points $C$–$F$ will all fail due to the inclusion of other edge nodes in their interpolation stencil (edge nodes cannot be used to reconstruct the values anywhere as they are the points we are attempting to reconstruct the value at in the first place). Thus, in this case, the reconstruction will fail to find the two required extrapolation points. Conceptually, this type of region is identical to a region in which two bodies intersect at an acute angle. In the region immediately adjacent to the intersection, there will be a set of edge points that cannot use the surface normal extrapolation technique.
Another situation in which the surface normal extrapolation reconstruction technique does not yield adequate results is at sharp corners in the body, as at airfoil trailing edges, etc. At a sharp edge the surface non-penetration condition cannot be applied because there is no well-defined body normal direction. This prevents the code from setting the direction of the velocity at that point on the surface such that it maintains the correct smooth flow reconstruction at the gridpoints near the sharp corner. This situation arises in this type of flow solver because applying a stagnation boundary condition at the sharp edge (the mathematically correct solution) results in poor flowfield development when it is interpolated back to the edge points where the boundary conditions are being set.

To correct for these problems, a modified method is used at the failed points that simply averages the surrounding points to force the flow to depart the body surface smoothly. Further development and extension of this technique is discussed in Section 3.7.7.

3.7.6.2 No-Slip Wall Results

First, for comparison with the no-slip-wall penalization methods, the surface normal extrapolation (SNE) technique was run using a no-slip-wall boundary condition. In these results a simple first-order extrapolation of all flow quantities) was used. Results are shown both with and without the use of the normal pressure derivative equation. Figure 3.22 shows the $C_l$ convergence time history. As with the penalization methods, neither the coarse nor the fine grids converge to a steady state solution, though the oscillations in this case are much smaller than those seen in the penalization methods. The use of the surface nor-
mal pressure equation has a strongly negative effect on the flowfield prediction in these cases. Figure 3.23 shows the $C_p$ distribution at the last timestep: while perhaps somewhat improved over the penalization cases, both pressure extrapolation techniques and both grid densities show substantial differences from the OVERFLOW prediction. Figure 3.24 shows the $C_p$ field around the airfoil: examining the pressure distribution in the region near the surface it can be seen that unlike the penalization cases which showed large-scale flow separation, the results for these cases appear more like small-scale numerical oscillations near the surface.
Figure 3.22: Lift coefficient vs. time for the no-slip wall SNE reconstruction cases. Dashed line shows nominal thin-arifoil result, $C_l = 0.33$
Figure 3.23: Pressure coefficient distribution for the no-slip wall SNE reconstruction case. Solid line is the inviscid OVERFLOW solution, symbols are the IBSEN solution.
Figure 3.24: Pressure field for the no-slip wall SNE reconstruction case.
3.7.6.3 Slip Wall Results

Unlike penalization methods, reconstruction techniques are capable of enforcing a slip-wall boundary condition by setting only the normal component of the surface momentum to zero (rather than the entire momentum). This is then interpolated back onto the edge points resulting in an approximate enforcement of the surface non-penetration condition in general, and an exact enforcement if the interpolation and extrapolation are designed specifically to work with the differentiation scheme. In this case the differentiation scheme is reduced to a first-order upwind scheme at the body surface, so when used with a linear extrapolation the non-penetration condition is satisfied. Higher-order techniques have been implemented and tested, but in general result in poor stability or worse surface pressure prediction results. Figure 3.25 shows the $C_l$ convergence for the coarse and fine grids: here, when used with the slip-wall condition, both the coarse and fine grid solutions finally converge to a steady-state value for the lift, though in both cases the value is overpredicted somewhat. Figure 3.26 shows the surface pressure at the final timestep for each grid: the pressure distribution matches the OVERFLOW solution quite well except at the leading edge: in this high-gradient region the linear extrapolation from the flowfield substantially overpredicts the surface pressure. Increasing the grid resolution improves the prediction somewhat. Switching to a higher-order surface treatment and/or including the normal momentum equation pressure extrapolation technique change the solution somewhat, but in all cases the leading-edge pressure is significantly overpredicted. Finally, Fig. 3.27 shows the converged pressure field, demonstrating a smooth result with no premature flow separation or evidence of numerical oscillations near the surface.
While the pressure prediction along most of the surface matches the inviscid OVERFLOW result quite closely, the surface pressure at the leading edge is dramatically overpredicted when using this method. This holds true for various extrapolation polynomials (linear extrapolation results are shown here), ranging from zeroth-order to fourth order (the highest order tested). Higher-order polynomial extrapolation is generally avoided for the simple reason that the polynomial is not necessarily monotonic, and the error in the extrapolation and the instability in the algorithm to minor changes in the flowfield causes non-physical behavior in regions of high flow gradients (such as the airfoil leading edge). In this case, using the linear extrapolation with two points along the normal vector, we see that for very coarse grids such as this one, the magnitude of the negative pressure peak is greatly overpredicted. As the grid is further refined the results get closer to the distribution predicted using OVERFLOW, but since the goal of this research is to develop a coarse-grid method, it appears the further study of this technique is required to achieve adequate surface pressure predictions.
Figure 3.25: Lift coefficient vs. time for the slip wall SNE reconstruction cases. Dashed line shows nominal thin-airfoil result, $C_l = 0.33$
Figure 3.26: Pressure coefficient distribution for the slip wall SNE reconstruction case. Solid line is the inviscid OVERFLOW solution, symbols are the IBSEN solution.
Figure 3.27: Pressure field for the slip wall SNE reconstruction case.
3.7.7 Reconstruction Method 2: Inverse Distance Weighted Average

Up to this point no previous technique has adequately addressed the needs of this research project: namely, stability on a low-resolution mesh along with reasonable accuracy in a compressible, inviscid flow solution procedure. The more complex methods faired somewhat better than the less complex methods when considering the very simple, smooth, non-degenerate airfoil case, but any attempts to use the aforementioned techniques on the types of grids required for this research on actual representative 3D geometries met with complete failure. Not a single method was able to produce a flow solution under the desired conditions. However, in the development of these implementations, a new technique was used to overcome various degeneracies that were encountered. That technique, originally only used to handle cases where the surface normal extrapolation technique (heretofore the most promising of the techniques) was unable to calculate an appropriate boundary condition, was extended to the entire immersed boundary and tested for efficacy. This new technique is referred to as the Inverse Distance-Weighted Average (IDWA) technique.

This technique stems directly from the requirement to use coarse grids, and is designed to be highly robust even in the event that some areas of the geometry are under-resolved by the volume mesh. Ideally the flow solver should simply “smooth over” those regions with the assumption that the users have put grid points in regions they are particularly interested in, and have deliberately under-resolved regions they are not concerned about. The previous methods not only yielded relatively poor coarse-grid results, but required substantial
special treatment of regions such as sharp corners or under-resolved geometry features. In particular, techniques requiring long-distance extrapolation from the flowfield required techniques to cope with regions where those methods were unable to obtain enough data for the extrapolation (see Section 3.7.6.1).

To correct these deficiencies, a very simple new technique was developed, based on an inverse distance weighted average of the surrounding flowfield. This essentially results in a smoothed zeroth-order extrapolation of the surface boundary condition. A small correction is applied to avoid stair-stepping, but in general the method is quite straightforward.

Figure 3.28 shows a schematic of the general averaging technique: the general idea is to find a set of points in the flowfield that are nearest to the point we are reconstructing, and to use a distance-weighted average to create a value at the edge point.

Mathematically the distance-weighted average interpolation technique requires the point being interpolated to lie within the convex hull formed by the
set of known data points, whereas in this case we are “interpolating” to a point that lies outside that hull. The “inverse distance-weighted average” technique employed here is therefore more akin to averaging to a point lying on the convex hull and then using that reconstructed value to perform a zeroth-order extrapolation to the surface point we are actually setting the data at. Once the point data has been reconstructed at the point using the IDWA technique, the next step is to apply the boundary condition itself, either no-slip or non-penetration.

To apply the boundary condition, the technique must begin by determining how close the point lies to the body surface. Next, the weighted-average distance from the point cloud to the same surface point is evaluated. These two quantities form a ratio that determines what percentage of the normal momentum to subtract to apply a “non-penetration” condition at the current gridpoint. For example, if the surface point is 0.25 units from the surface, and the average distance of the point cloud to that surface point is 0.75, then 0.25/0.75, or 1/3, of the normal momentum should remain, resulting in 2/3 of the normal momentum being subtracted from the point. This helps to prevent a “stair-stepping” approach that occurs when applying the zero normal momentum boundary condition on points that do not actually lie on the surface. The technique allows a smooth gradation in the velocity, even though the locations the boundary condition is applied at jump as the straight lines of the grid do not exactly follow the curved surface of the body.

When choosing the points to use in the averaging, it is important that a point lying on the opposite side of a split boundary not be incorporated into the solution procedure (though in rare instances when it occurs the averaging process tends to reduce its influence and at least results in a stable flow solution). To
accomplish this a line segment between the nearest surface position and the potential field point is tested for intersection with the body: if it intersects, that point is discarded.

This technique has the advantage of requiring no special cases for sharp corners, surface degeneracies, etc. In addition, it cannot yield a value for the flow variables that is larger or smaller than any individual component, so is specifically designed to prevent sharp discontinuities from appearing due to the action of the immersed boundary condition. None of the flow quantities calculated using this technique can be local extrema in the flow: while not strictly physically accurate, this represents an appropriate compromise to yield the necessary flow solution stability.

3.7.7.1 No-Slip Wall Results

First, for comparison purposes with previous analyses the IDWA method was first run using a no-slip wall boundary condition. Figures 3.29a and 3.29b show the convergence history of the coefficient of lift. As with the previous no-slip wall results, the flow solution does not converge to a steady-state solution at these grid densities, instead exhibiting very large aperiodic force oscillations. Figures 3.30a and 3.30b show the $C_p$ distribution at the last computed timestep, again demonstrating the very poor results achieved with the high-Reynolds’ Number approximation. Finally, Figs. 3.31a and 3.31b show the flowfield results, again demonstrating very large aperiodic shedding phenomena. Clearly this technique is far worse than any of the others when used with the no-slip wall boundary condition.
Figure 3.29: Lift coefficient vs. time for the no-slip wall IDWA reconstruction cases. Dashed line shows nominal thin-arifoil result, $C_l = 0.33$. 
Figure 3.30: Pressure coefficient distribution for the no-slip wall IDWA reconstruction case. Solid line is the inviscid OVERFLOW solution, symbols are the IBSEN solution.
Figure 3.31: Pressure field for the no-slip wall IDWA reconstruction case.
3.7.7.2 Slip Wall Results

Figures 3.32a and 3.32b show the lift convergence for the slip-wall boundary condition case. The IDWA method shows substantial improvement over the other techniques when used with coarse grids. The use of the slip wall boundary condition results in a stable, physically-realistic flow solution when used with the IDWA technique. Figures 3.33a and 3.33b show the $C_p$ distribution at the last computed timestep, this time demonstrating very good results achieved with correct physical approximation of a slip-wall boundary condition, and showing an improved pressure prediction at the leading edge when compared to the results from the surface normal extrapolation technique. This advantage appears to decrease as the grid resolution is increased, but both results are very good, demonstrating that it may be possible to use this technique to enable the use of very coarse grids with the immersed boundary technique. Figures 3.34a and 3.34b show the pressure contours around the airfoil, as expected showing a smooth, unseparated flowfield.

These results are very promising when applied to coarse grids, providing both a reasonable surface pressure and volume flowfield at the grid densities considered here. However, since surface pressure calculations are not required on the fuselage, it may be possible to extend the method to even coarser grids, with the eventual goal being to simply immerse the fuselage in a pre-existing isolated rotor mesh system with no modifications, ultimately allowing the inclusion of the influence of the fuselage on the acoustic predictions for no additional cost over the original isolated rotor simulations. Toward this end, Section 3.7.7.3 examines a series of coarser grids for the same two-dimensional case considered here. However, it is suggested by these results that this technique is capable
of providing a simple way to analyze the flowfield and acoustics of complex heavy-lift rotorcraft on computationally-feasible immersed-boundary grids.
Figure 3.32: Lift coefficient vs. time for the slip wall IDWA reconstruction cases. Dashed line shows nominal thin-arifoil result, $C_l = 0.33$
Figure 3.33: Pressure coefficient distribution for the slip wall IDWA reconstruction case. Solid line is the inviscid OVERFLOW solution, symbols are the IBSEN solution.
Figure 3.34: Pressure field for the slip wall IDWA reconstruction case.
3.7.7.3 Coarse Grid Analysis

The inverse distance weighted average technique shows excellent results for a simple airfoil geometry in two dimensions at both of the grid resolutions analyzed for the other methods (when used with a slip-wall boundary condition). Due to its excellent stability characteristics, the IDWA method was then able to run a wider range of grid densities, ranging from a very fine $\Delta x = 0.0025$ chords to an extremely coarse $\Delta x = 0.08$ chords. Figures 3.35a through 3.35g show the pressure field for each successive doubling of $\Delta x$, as well as a comparison to a relatively coarse body-fitted mesh. In particular note that not only is the IDWA method stable for even the coarsest grid, but the solution values found are not unreasonable considering the grid fidelity. No other immersed boundary method analyzed in this research was able to achieve these results.

This result is particularly important when considering under-resolved geometry features. While it is unlikely that a user would ever run a complete case with a fuselage at this resolution, it is possible that the fuselage geometry used may have features that are under-resolved to the extent studied here. Because one of the key advantages of the immersed boundary technique is the simplicity of its user inputs, retaining that simplicity even in the face of an unnecessarily detailed fuselage model can save significant time when setting up a new case, eliminating the necessity for time-consuming and costly geometry cleanup. This case demonstrates that even if the fuselage has complex protuberances (pylons, winglets, or even features such as pitot probes and antennae) the flow solver will a) be able to execute the run, and b) will not exhibit any extreme behavior near the under-resolved features. This also allows the user to set up a simple coarse case at the beginning of the analysis, and to progressively refine
the fuselage grid as needed and as computational resources allow, without ever having to switch to a different fuselage model.
Figure 3.35: Comparison of pressure fields for various grid densities using the IDWA immersed boundary technique.
3.7.7.4 Stationary 2D NACA 0009 angle of attack sweep

The next IDWA immersed boundary validation test case is a static angle of attack sweep at a freestream Mach number of 0.2 at standard sea level conditions. To change the angle of attack of the airfoil the inner grid and airfoil body were rotated relative to the stationary outer grids and uniform incoming freestream flow. Fig. 3.36 shows the outer domains of each grid in this 6-grid overset system, including an inner grid rotated to three degrees angle of attack.

Figure 3.37 shows the innermost grid along with an outline of the actual geometry: this is a fairly fine inner grid designed to allow accurate prediction of the surface pressure, with approximately 23 points across the thickness of the airfoil and 125 points chordwise. The cell aspect ratio in this case is two.

Figure 3.38 shows the pressure contours for the three degree angle of attack case for IBSEN and Fig. 3.39 shows the results for OVERFLOW run in inviscid mode. The results show very good qualitative and quantitative agreement, though close examination near the surface shows some differences due both to the way the immersed boundary code constructs the boundary conditions as well as to the coarse grid used in these OVERFLOW simulations.

Figure 3.36: Example of the overset grid system used in the NACA 0009 angle of attack sweep.
Figure 3.37: Inner grid showing level of surface resolution.

Figure 3.40 shows the IBSEN results for the momentum vectors at the leading edge of the airfoil as well as the contours of velocity magnitude. No “stair-stepping” effects are evident in the computation and the field varies smoothly even in the vicinity of the stagnation point.

Figure 3.41 shows a comparison of the surface pressure predicted by IBSEN and OVERFLOW. A small discrepancy is apparent on the upper surface at the leading edge, but this small difference does not strongly affect the integrated lift calculations, nor is it expected that a difference such as this will have a strong
Figure 3.38: Contours of pressure coefficient calculated by IBSEN for a NACA 0009 airfoil at 3 degrees angle of attack.

Figure 3.39: Contours of pressure coefficient calculated by OVERFLOW for a NACA 0009 airfoil at 3 degrees angle of attack (inviscid result).
Figure 3.40: Momentum vectors and velocity magnitude contours at the leading edge of a NACA 0009 at 3 degrees angle of attack. IBSEN prediction.

Figure 3.42 shows a comparison of the calculated and experimental lift curves for the NACA 0009. The experimental results are shown for Re=360,000 and the IBSEN results are inviscid with a slip-wall boundary condition. IBSEN shows excellent correlation with the experimental results through five degrees angle of attack, but begins to diverge as IBSEN shows premature stall behavior. The basic effect appears to be that, with the current immersed boundary condition implementation, IBSEN simulates flow at a grid-dependent Reynolds’ number that is quite low for the grid densities under consideration. In regions where flow separation is expected (aft of a rotorcraft fuselage, for example) it
Figure 3.41: Comparison of IBSEN and OVERFLOW surface pressure predictions for a NACA 0009 airfoil at 3 degrees angle of attack.

is clear that this technique will not yield high-quality results. However, for the purposes of this research (blade loading calculations for input into acoustics propagation simulations) this is not expected to have a strong influence on the results.
Figure 3.42: Coefficient of lift versus angle of attack for IBSEN NACA 0009 run, compared to experimental results at Re=360,000.

3.7.7.5 Three-Dimensional Analysis

In the preceding sections only two-dimensional cases were considered; however, the eventual use for the immersed boundary solver is to capture the flow around three-dimensional rotorcraft fuselages to allow appropriate calculation of the fuselage influence on the rotor inflow (and therefore its acoustics). To evaluate the efficacy of the new IDWA technique on a rotor fuselage, a simple representative geometry was chosen: the ROBIN (Rotor Body INteraction) geometry. This is a smooth, thin, analytically-defined geometry representative of a class of small rotorcraft fuselages, shown in Figure 3.43.

Because the aim of this research is to approximate the influence of the fuselage on the rotor inflow, neglecting the surface pressure accuracy (except insofar
as it must be accurate enough to reproduce the correct flow at the rotor location), a variety of very coarse grid densities were examined, and compared to the flowfield results from an OVERFLOW simulation using a body-fitted surface mesh system.

Figure 3.44 shows the three grid systems analyzed in this case: the white area shows where the cells interior to the ROBIN fuselage were removed (note that the immersed boundary technique uses the actual location and orientation of the fuselage surface in its analysis, the white area is shown only for reference purposes to illustrate the fidelity of each mesh). Figure 3.45 shows a comparison of the pressure field results on a plane down the centerline of the fuselage. The solid black lines show several levels of pressure contours calculated by an inviscid OVERFLOW run using a body-fitted mesh, while the dashed and dotted colored lines show the same pressure contours as calculated by IBSEN for the three meshes described above. Note again that even the coarsest of the meshes is stable, and although its results are beginning to diverge from the OVERFLOW solution it is not unreasonable. In addition, both of the other two cases show excellent agreement with the body-fitted mesh solution. The fine and medium grid densities used in this simulation are representative of the expected density of the rotor level-one mesh (that is, the innermost Cartesian mesh enclosing the
body-fitted blade meshes). The coarse grid density is representative of a level two or level three block, which the lower part of a rotorcraft fuselage may encounter during an immersed boundary run. This case demonstrates that for a practical rotor geometry, the expected grid densities of a normal isolated rotor grid are adequate to model the effects of the fuselage on the flowfield in the immediate vicinity of the rotor, and that even the coarser off-body meshes will yield stable, physically-realistic results.

3.7.8 Computational Cost

One of the basic goals of this research was to develop a technique that would allow the inclusion of a fuselage model in a grid originally designed for isolated rotor simulations, without requiring modification of that grid. By eliminating any increase in the number of gridpoints, computational cost associated with including the fuselage in the simulation may greatly decreased. The extent of the time-savings depends in large part on whether the fuselage is immersed in a grid moving relative to the geometry. In such cases, the grid must be re-analyzed for intersection with the fuselage at every timestep, significantly slowing down the simulation. Although IBSEN supports this case, all research presented here focuses on simulations where moving grids do not intersect the immersed boundary. In this case, the vast majority of the computational cost of the method is focused in the initialization stages of the case, and the increase in computational cost of each subsequent timestep is negligible.
Figure 3.44: Immersed Boundary near-body grids used in ROBIN analysis.
Figure 3.45: Pressure contours on a plane through the center of the ROBIN fuselage, $M_\infty = 0.2$. 
Vortex Interaction Strength

Preservation

In cases where the vortices generated by upstream structures (particularly previous rotor blades) move close to succeeding blades or other noise-generating surfaces, the unsteady pressure caused by these interactions can become a dominant noise source. It is therefore important that a flow solver designed specifically to generate inputs into acoustic propagation codes to be able to predict the location and strength of these interactions. With conventional Eulerian flow solvers this is extremely challenging due to the tendency of such methods to artificially numerically diffuse the generated vortices much more quickly than the vortices diffuse in experiments or flight conditions. Very large numbers of grid cells are sometimes used to counteract this effect, but even with relatively fine volume grids the vortices still tend to “smear out,” reducing both their peak strength and the time derivative of the induced pressure on the blade surface (a quantity critical to acoustic predictions).

Furthermore, the code developed in this project has been specifically de-
signed to minimize the number of gridpoints required in the calculation through the use of the newly-developed coarse grid Immersed Boundary presented in the previous chapter. Allowing the number of gridpoints to increase dramatically in order to fully capture the rotor tip vortices runs counter to the goal of the project.

The next major contribution of this research is the development of a new technique for preserving the strength of vortex-structure interactions for input into the final acoustics calculation. This new method allows fast, approximate prediction of BVI noise by estimating the position of the vortex and providing a model for the vortex pressure contribution.

4.1 Necessity of Vortex Interaction Strength Preservation

A vortex passing near to a blade induces a large, localized velocity jump at the blade surface, resulting in a highly-impulsive pressure fluctuation as the vortex sweeps past the rotating blade. Since the generated acoustic pressure wave is a function of both the loading and the time derivative of the loading, this impulsive spike in pressure causes a very large pulse to propagate out from the interaction location. The key to predicting BVI noise is to reproduce the impulsive loading seen on the blade surface. This can be done via direct simulation of the vortex, or through an approximate method of predicting the loading: the aim of this research is to develop a suitable approximate method.
4.1.1 Modern High-Fidelity CFD

Other research to date into improved BVI prediction in CFD has focused on directly improving the CFD prediction of the vortices by using sophisticated high-order numerical methods as well as increased numbers of grid points. While progress in this field has been steady, it comes at the cost of dramatic increases in computational requirements.

For example, in Ref. 10, the HELIOS flow solver was used to examine the HART II experiment and evaluate the present state of the art in high-fidelity CFD predictions of tip vortex strength. In this research the authors analyzed a number of different grid densities and refinement techniques with the aim of inserting enough grid points across the vortex core region to maintain its integrity throughout the simulation. What they found was that even with advanced adaptive grid techniques their reproduction of the tip vortex strength was extremely limited. However, their predictions of the vortex positions were excellent.

Figure 4.1 shows a comparison of the HART II experimental vortex locations (shown as red spheres in the images) and the predicted flowfield from four different HELIOS grids. The positions of the vortices at all locations and for all grids show excellent agreement with the experimental results, demonstrating that modern CFD methods are capable of predicting the position of tip vortices as they evolve.

However, Fig. 4.2 shows a comparison of the measured and predicted strengths using the best available computational grid, demonstrating that although the vortex positions agree well with the experiment, their strength is severely underpredicted.
The grid labelled “8L-AMR-Q” was designed such that it had four gridpoints across the predicted vortex core size, and was developed using adaptive mesh refinement based on the Q criterion for identifying rotor tip vortex structures. This mesh had approximately 202 million gridpoints. The authors suggest that a grid with eight points across the core would improve the predictions, but would require 1.8 billion gridpoints in the AMR case, and 5.0 billion in a standard, non-adaptive mesh.

The aim of the present research is to demonstrate a new approach to the prediction of blade vortex interactions when the ultimate goal is acoustic predictions. In it, rather than attempting to directly capture the strength and motion of the vortex using Eulerian techniques (a simulation technique currently computationally infeasible in a design setting due to the huge number of gridpoints required), a combined Eulerian-Lagrangian approach is taken where the Eulerian flow solver is used for the bulk flow and vortex formation predictions, but the long-term convection of the vortex is handled in a Lagrangian manner.
Figure 11. Vortex wake positions computed with four different off-body grids (measured data shown by the red spheres) in the longitudinal plane at $y=1.4\text{m}$: a) 6L-fixed, b) 7L-fixed, c) L8-AMR-v, d) 8L-AMR-Q.

$\Psi = 20^\circ$

a) 6L-fixed

$\Psi = 70^\circ$

b) 7L-fixed

c) 8L-AMR-v

d) 8L-AMR-Q

Figure 4.1: HELIOS vortex position prediction for four different grids (Figure 11 from Ref. 10). Red spheres show experimental position measurements.
Figure 14. Measured PIV vorticity field for wake ages from 5.3° to 45.3° in a vortex roll up (temporal investigation, positions 17a-h).

(a) Experiment.

(b) HELIOS.

Figure 4.2: Comparison of experimental and predicted vortex strengths (Figures 14 and 15 from Ref. 10).
4.2 Mathematical Model of Vortex and Convection

In 2D potential flow theory a the velocity field induced by a point vortex is given in cylindrical coordinates by

\[ V_{\theta} = -\frac{\Gamma}{2\pi r}, \quad V_r = 0 \] (4.1)

where \( \Gamma \) is the circulation. From the superposition principle, potential flow theory predicts that a vortex will convect with the local velocity field forever, its strength unchanging in time. Note that potential flow theory predicts a singularity at the vortex center, where the velocity goes to \( \infty \). A real “2D” vortex behaves slightly differently: first, a real vortex has a finite velocity at the core, where due to viscous effects the fluid rotates about the center in a manner similar to a rigid body. Second, that vortex core grows (slowly) over time to due viscous effects.

The precise velocity field of a real vortex is complicated (the core region has very strong viscous effects, for example), but one model of a finite-core, vortex frequently used in rotorcraft flowfield analysis is the Vatistas series of vortex models, whose form is:

\[ V_{\theta}(r) = \frac{\Gamma_v}{2\pi} \frac{r}{(r_c^{2n} + r^{2n})^{1/n}} \] (4.2)

where \( r_c \) is the viscous core radius and \( n = 1, 2 \) selects the velocity profile. For this research \( n = 1 \) was chosen, resulting in a “Scully model” vortex. Figure 4.3 shows a comparison of the potential vortex and the Scully model vortex velocity profiles. The fundamental problem of BVI noise prediction is therefore to accu-
Figure 4.3: Vortex velocity profiles for a potential and a Scully-model vortex, $\Gamma = 1.0, r_c = 0.1$.

deliberately convect this vortex downstream with no non-physical dissipation. In an Euler flow solver with no viscous terms, the vortex should convect through a uniform freestream undisturbed. Any decrease in the vortex strength is due to numerical diffusion inherent in the discretized equations.

4.3 Description of New Method

The keys to the new method presented here are that:

1. The Eulerian flowfield solution accurately captures large-scale bulk flow effects and therefore accurately predicts the position of the rotor tip vortex.

2. The solver is incapable of maintaining the high-gradient rotor tip vortex,
diffusing its energy into the bulk flow far more quickly than is observed in reality.

3. The acoustics of a blade-vortex interaction are very strongly influenced by both the magnitude and the time derivative of the blade surface pressure fluctuation.

A new technique is therefore developed that can approximately reconstruct the acoustics of blade-vortex interactions without being cost-prohibitive, particularly in a design environment. An Eulerian flow solver is desirable for its ability to accurately model complex interactional aerodynamics, so what is needed is a technique for including the important features of the vortex-structure interactions that works within an Eulerian flow solution framework.

The general idea is to carry along extra information during the flow solution procedure related to the original undissipated strength of the vortex. When the pressure on the blade surface is calculated for the acoustics output, an additional calculation is performed to augment the calculated surface loading with an induced component due to the vortices. This augmentation is calculated by determining how much the vortex has dissipated and then artificially introducing a Scully vortex whose strength is equal to the lost circulation. This procedure double-counts the energy from the vortex in the sense that the flow solver has typically not “lost” the contribution of the vortex, but merely spread it out in a non-physical manner. However, the new method enables the recreation of the high pressure derivatives critical for predicting BVI noise. The effect of the double counting can be mitigated by using only the vortex filaments very close to the blade to augment the loading, and by ensuring that they are only used
when the flow solver has actually allowed the vortex to dissipate (though in most cases it has).

In a procedure similar to the vortex core tracking method used in vortical flow visualizations, “massless particles” are periodically released into the flow from the rotor blade tips. The code takes as user input a small circular plane on which the vortex particles are released. The code calculates the vorticity on this plane and chooses the position of highest vorticity as the release point. This calculation is performed at each particle release step, and for each rotor blade. The frequency of particle release is a user input, typically on the order of one particle released per degree of rotor revolution.

When the particle is first released, a series of quantities are calculated. First, the expected peak vortex strength after vortex roll-up is determined by examining the circulation on the rotor blade at the release time. The maximum bound circulation on the blade at release time is stored by each particle and may be used to approximate the expected final vortex strength (Ref. 134). In addition, the actual local circulation as predicted by the CFD solver is calculated and stored for both the initial release time as well as at each output time, allowing visualization of the circulation as a function of time. The vortex strength is calculated by numerically integrating the velocity on a circular path perpendicular to the filament to calculate the circulation. The radius of this path is a user input, typically $10r_c$ where $r_c$ is the user-input viscous core radius (see Section 4.4 for details).

These massless particles are then convected along the local velocity vector at each timestep. Taken together the collection of particles traces out the tip vortex trajectory as a function of time, as predicted by the Euler equations of
fluid dynamics. Unlike a classical Lagrangian approach, the particles do not influence the flowfield development, but serve only as markers for the vortex location in the Eulerian field, and containers for the expected vortex strength. Although the Eulerian prediction of the vortex strength diminishes rapidly in time as the vortex core expands rapidly (a characteristic of this type of flow solver), the particles retain a “memory” of the original strength. This strength is used in a later step to calculate the impulsive blade loading. The simulation complexity is not strongly influenced by the number of particles, as it would be in a full Biot-Savart influence calculation. The Euler equations solved on the Eulerian field are responsible solely for capturing the bulk fluid dynamics of the simulation, with the vortex particles used only in the blade surface pressure predictions. The particle convection term calculation is orders of magnitude faster than the Eulerian CFD calculation and has negligible affect on the code’s runtime.

During those blade loading predictions, the first step is to extrapolate the pressure from the Eulerian field onto the blades, as in a conventional simulation. Using the chordwise compact aeroacoustic approximation the pressure is integrated to yield loading vectors distributed along the blade’s quarter-chord line. In a conventional simulation this loading is then passed to the acoustic propagation code and the acoustic pressure is calculated. In the technique developed in this research, an additional step is employed, however.

Once the initial loading from the CFD is determined, the particles are analyzed in a seven-step process:

1. The current circulation at the particle locations is calculated by numerical integration.
2. At each blade quarter-chord point, the set of particles that lie within a user-specified radius (typically a few blade chords) is looped over:

3. These particles form a linked series of vortex filaments that is used to calculate an induced velocity at the quarter-chord position.

4. The representative local velocity vector is calculated from the CFD flowfield.

5. Assuming $c_{L_{\alpha}} = 2\pi$, an approximate lift vector is calculated based on the baseline calculated velocity vector.

6. Again assuming a $2\pi$ lift curve slope another approximate lift vector is calculated, this time adding in the induced velocity.

7. The difference between the two approximate lift vectors is added to the lift vector calculated directly from the CFD.

At each adjusted point this will result in a small over-prediction of the loading due to a "double-counting" of the vortices' influence: once from the Eulerian field prediction, which will result in a diffused, non-impulsive loading, and once from the contribution due to the vortex particles. However, due to the nature of the acoustics predictions, this small pressure overprediction is relatively insignificant. The acoustics of a BVI are very strongly dependent on the magnitude and the time derivative of the surface pressure. Despite the potential for a small pressure magnitude overprediction, the time derivative of pressure is captured much more accurately using this system, so it is believed that this small potential overprediction is superior to the dramatic underprediction of the BVI noise evidenced without the particles.
4.3.1 Implementation Details

When the particle is first released, the local circulation is calculated by evaluating the line integral of velocity on a circle of user-specified radius oriented on a plane perpendicular to the vortex. At each timestep the particles’ position and velocity are updated using simple explicit timestepping and trilinear interpolation, respectively. Because the particle motion is calculated using this numerical method, small errors in the particle position will accumulate over time. However, because the timestep size is typically on the order of 1/10th degree of rotor revolution per step, the particle convection is expected to be adequate in most cases. The most significant departure from the physical solution will occur in regions of vortex-vortex interaction: because the flow solver is expected to almost completely dissipate the vortices into the bulk flow, the particle trajectories will not reflect these types of interactions correctly.

The actual calculations take place in three distinct steps, each of which is independent of the others and in general occur at different timesteps:
Vortex particle creation: Vortex particles are created at user-specified positions and timesteps. The creation involves calculating the initial circulation to be carried by the particle by performing the line integral of velocity:

$$\Gamma = \oint_C \vec{V} \cdot dl$$  \hspace{1cm} (4.3)

This integration is performed on a circle of user-specified radius in a plane oriented perpendicular to the local vorticity vector. In the current implementation the circle is divided into 18 segments. The endpoints are then interpolated to from the flow solution and used to perform the simple nu-
merical integration. Finally, this initial circulation is then stored with the particle.

**Particle convection:** Every timestep vortex particle positions are updated by integrating them along the local velocity vector. Fig 4.4 shows a schematic of the general idea using a first-order integration scheme. Runge-Kutta integration is used to be consistent with the rest of the flowfield. Figure 4.5 shows how a set of periodically-released vortex particles could track along with the local flowfield to calculate the position of the tip vortex.

**Circulation update:** During the course of the run the local circulation at the current particle position is calculated in the same manner as the original circulation, allowing the tracking of any vortex roll-up that the solver captures, as well as a determination of how much the particle has dissipated over the course of the simulation.

**Surface loading calculation:** At a user-specified interval the surface loading is calculated. All of the particles within some distance of the position to calculate the loading at are gathered by the main process and formed into ordered lists representing the vortex strands. Using the Biot-Savart law, this strand is integrated along, yielding induced velocities on the quarter-chord locations. These velocities are then used to calculate the augmented surface loading. Figure 4.6 shows a schematic of the vortex particles used to calculate the loading at a given location on the blade surface (the green point). All vortices within a specified radius of the point are linked together to form a vortex filament. Each filament segment is treated as a line vortex with a strength given by the average of the two endpoints.
4.3.2 Vortex-Induced Velocity Field

In three-dimensional inviscid, incompressible flow the induced velocity field due to an infinitesimal-length vortex filament can be evaluated using the Biot-Savart law:

\[ d\vec{V} = \frac{\Gamma}{4\pi} \frac{d\vec{l} \times \vec{r}}{|\vec{r}|^3} \]  

(4.4)

where \(d\vec{l}\) is the infinitesimal filament and \(\vec{r}\) is the radius vector from the filament to the point in space where the velocity is being evaluated. Implemented numerically the infinitesimal filament is typically modeled by a short straight-line filament. The helical vortex shed from a helicopter rotor is then formed from hundreds or thousands of these straight-line filaments, connected end-to-end, each with their own strength, and the induced velocity at a point is calculated by summing the contributions from each vortex filament. Much of classical rotorcraft analysis relies on using these filaments to approximate the rotor wake quickly and efficiently. Adapting the Biot-Savart law to the Scully model vortex, we then have, for each filament:

\[ V_\theta(r) = \frac{\Gamma_v}{2\pi} \frac{h}{(r_c^{2n} + h^{2n})^{1/n}} (\cos \theta_1 - \cos \theta_2) \]  

(4.5)

where the included angles \(\theta_1\) and \(\theta_2\) describe the position of this finite-length segment with respect to the point the velocity is being induced at.

4.4 Calculation of Vortex Strength

Determining the strength of the shed vortices is an important step in the use of this new technique. Several methods are available: first, the circulation may be
calculated directly from the CFD prediction at some time after it is shed, and then assumed constant. Second, the circulation can be continuously calculated from the CFD prediction, and the largest value encountered can be used as the particle circulation (therefore theoretically accounting for the vortex roll-up, but requiring the CFD to adequately predict the circulation through that rollup process). A third technique is to approach the problem more akin to the technique used by conventional particle methods, and use the bound circulation on the lifting surface to predict the expected vortex strength. All three techniques are available in IBSEN: the strengths and weaknesses are discussed next.

4.4.1 Direct calculation of circulation

The circulation strength of the vortex can be calculated via the definition of circulation:

\[ \Gamma = \oint_C \vec{V} \cdot \, dl \]  

(4.6)

For a true potential vortex \( \Gamma \) is a constant for any closed integration path enclosing the vortex core. For a vortex as modeled by the Scully equation the calculated circulation approaches \( \Gamma \) as the integration radius is increased. Taking a line integral around this vortex on a circle at radius \( r \):

\[ \Gamma = \oint_C \left( \frac{\Gamma_v}{2\pi} \right) \left( \frac{r}{r_c^2 + r^2} \right) \cdot \, dl \]  

(4.7)

Since we are integrating over a circle, and the velocity is the tangential velocity by construction, all of the quantities inside the integral are constant:

\[ \Gamma = \left( \frac{\Gamma_v}{2\pi} \right) \left( \frac{r}{r_c^2 + r^2} \right) \oint_C \, dl \]  

(4.8)
Integrating around the circumference of the circle,

$$\Gamma = (2\pi r) \left( \frac{\Gamma_v}{2\pi} \right) \left( \frac{r}{r_c^2 + r^2} \right)$$

or

$$\Gamma = \Gamma_v \left( \frac{r^2}{r_c^2 + r^2} \right)$$

Thus, when $r^2 \gg r_c^2$, the equation reduces to $\Gamma = \Gamma_v$. It is therefore important in the numerical integration of circulation that a large enough integration radius is chosen, allowing the integral to capture the full strength of the vortex. In practice, however, there is a limit to how far away the integral may be taken, due both to the complexity of the flowfield and to the reduced grid density available farther from the rotor blade (our vortex-generating surface). For this research an integration radius of ten times the expected core radius was used throughout: this is typically on the order of one blade chord, and typically results in the integration circle being entirely contained within the near-body grids.

For some cases, particularly those with high-fidelity near-body grids, this technique allows the particle method to rely entirely on the CFD prediction of the vortex formation, only approximating the final position of the vortex, but allowing the CFD to predict its initial strength. The downside of simply using a single integration to calculate the strength is that it expects the vortex to be fully formed at the point the integration is formed, resulting in an integration circle that must be place far enough downstream of the generation to allow the vortex to roll up (typically on the order of 2-3 chords). This additional distance makes the prediction of the necessary vortex release position more error prone, as a larger potential seeding surface must be examined.
4.4.2 Direct calculation of circulation and roll-up

If, instead of only calculating the circulation once, at the particle release time the circulation is recalculated at every timestep, it is theoretically possible to allow the particle method to track the CFD prediction of the vortex roll-up process, and to use the circulation from the fully-formed vortex. This follows the technique described in the previous section, but is repeated for all particles at all timesteps. The peak circulation that each particle encounters throughout its evolution is then used as the circulation in the acoustic prediction. This allows the seed surface to be placed very close to the trailing edge of the rotor blade, enabling better accuracy in the particle placement. Its downside is that adequate grid resolution must be maintained throughout the vortex roll-up process, and the flowfield must be relatively stable during that process: if the flowfield is strongly influenced by other, non-tip-vortex elements, the circulation prediction may not be representative of the circulation due to the vortex alone (which is the desired quantity).

4.4.3 Approximation of circulation using bound circulation

The final method for calculating the vortex circulation is to borrow a technique from conventional particle methods, and base the circulation on the bound circulation of the surface the vortex is trailing from (in this case a rotor blade). This method depends only on the flowfield on the surface of the blade. Unlike conventional particle methods, where various lower-fidelity methods may be used to approximate this bound circulation (airfoil lookup tables, etc.), in our case the circulation can be evaluated directly from the CFD via the definition of
circulation

\[ \Gamma = \oint_{C} \vec{V} \cdot dl \quad (4.11) \]

This time instead of performing the integration on a circle around the vortex filament, the integration is performed on each of the airfoil sections that form the blade surface. What results is a map of circulation versus spanwise location along the blade. The conventional approximation from other single-trailer vortex particle methods is then used, where the shed vorticity is assumed to coalesce into a single strong tip vortex of strength \( \Gamma_{\text{max}} \). This technique does not rely on the flowfield any distance from the blade surface, so is less sensitive to a rapid decrease in grid resolution: as such it is most appropriate for very coarse grid simulations where the vortex roll-up is not expected to be captured by the CFD simulation. The downside is that it tends to overpredict the tip vortex strength by assuming a complete inclusion of all generated vorticity into a single trailer.

### 4.5 Vortex Formation, Growth, and Destruction

A vortex does not form instantly upon release from its generating surface: rather, a continuous sheet of vorticity is shed from the entire trailing edge of the wing (or rotor blade), and over time that sheet coalesces, or “rolls up,” into shield tip vortices. Experiments have shown that this coalescence is quite rapid, requiring only 2-3 chordlengths of downstream convection to give a fully-formed tip vortex, but it is not instantaneous (see, e.g. Ref. 134 Section 10.6). As such, a method that requires knowledge of the vortex strength must account for this period, either by waiting until the vortex is formed to calculate the circulation
(as in the method presented in Section 4.4.1), continuously re-evaluating the vortex strength (as presented in Section 4.4.2), or by including an explicit model for the roll-up procedure (necessary when using the bound-circulation method presented in Section 4.4.3).

In this research a very simple linear model of vortex formation is used in conjunction with the bound circulation calculation method. This method is designed primarily to prevent the vortex from inducting a velocity on the blade that generated it. An additional user input, the “roll-up time,” is provided to the code, which then calculates the difference between that time and the release time of the particle and scales the vortex strength accordingly:

$$\Gamma = \begin{cases} 
\Gamma_{\text{bound}} \left( \frac{t - t_{\text{release}}}{t_{\text{roll-up}}} \right) & : t - t_{\text{release}} < t_{\text{roll-up}} \\
\Gamma_{\text{bound}} & : t - t_{\text{release}} \geq t_{\text{roll-up}} 
\end{cases}$$  \hspace{1cm} (4.12)

Once the vortex is formed, its core gradually expands over time due to viscous effects. While the effect is greatly amplified in present numerical predictions and results in the mis-predictions that the current method addresses, the core of a physical vortex does grow over time. Figure 4.7 shows a plot of various experiments showing this effect. To ensure that any direct blade-vortex interactions that occur (those interactions where the blade actually interacts with the core region and near-core region) are appropriately representative of the real expected interactions, IBSEN includes a model of this core growth. User inputs to the model are the initial vortex core size, and the time required for the first core size doubling to occur. This is used to calculate the coefficients $\nu$ and $t_0$ in
the core growth model

\[ r_c = \sqrt{v(t - t_0)} \]  

(4.13)

such that the core radius \( r_c \) at time \( t = t_0 \) is the input initial core size, and that at time \( t = t_0 + t_{\text{doubling}} \), the core radius \( r_c = 2r_{c,\text{initial}} \).

Finally, there are circumstances under which the standard core growth is short-circuited and the core grows extremely rapidly, effectively eliminating a section of the vortex. This process is referred to in the literature as “core bursting” and can occur when the vortex filament encounters a physical obstacle, such as when a rotor blade passes directly through a section of the trailed vortex. While experimental results are extremely sparse for this process, and the process itself highly complex and dependent on highly-detailed knowledge of the flowfield, the general idea of core bursting has been roughly incorporated into IBSEN in order to handle cases with widely divergent particle paths, typically seen when one particle passes above the blade and the next passes below it, resulting in a discontinuity in the filament. A very simple model of this process
was implemented: if a filament is found to intersect a solid object at any time, a simple rapid linear decay is activated such that within a small amount of time the circulation strength of the filament is reduced to zero, preventing its use in the acoustics simulations. This ensures that no non-physical discontinuities are introduced into the acoustic prediction due to these divergent particle paths. While not a perfect model of the core bursting seen in reality, it is adequate for this purpose.

4.6 Particle Tracking Demonstration Cases

To demonstrate the basic capabilities and features of the particle tracking method, three cases are presented here. First, a ring vortex artificially inserted into a uniform freestream and convected first using a modified free-vortex solver and then using IBSEN and the particle tracking technique. Second, a simplified two-dimensional vortex convecting past an airfoil. Third, a three-dimensional vortex generated by a finite-span wing and allowed to convect downstream.

4.6.1 Ring Vortex

In this case, a uniform flow was created and a finite-core Scully-model vortex ring was explicitly inserted into the flow. Two techniques were then used to convect the vortex. First, as a proof-of-concept demonstration performed without using IBSEN, a simple flow solver was developed in which the ring was maintained as a set of Lagrangian free-vortex filaments (36 straight-line filaments were used to approximate the circle). The Biot-Savart law was used to calculate
the velocity at each filament endpoint, which were then advanced in time using a second-order Runge-Kutta timestepping scheme, as in a free-vortex solver. In addition to their mutual induction operation, the vortex filaments were used to calculate the velocity field on a uniformly-distributed Cartesian grid. This velocity field was then used with a second-order Runge-Kutta scheme to convect a set of massless particles in the local linearly-interpolated velocity direction, following the same technique that will later be demonstrated in IBSEN. The aim of this case is to demonstrate the accurate convection of the massless particles in an ideal vortex flowfield, independent of the expected diffusive effects expected when used within a true Eulerian flow solver.

Figure 4.8a shows the results from the baseline free-vortex/field interpolation solution. Because the vortex strength is undiminished during the course of the run, the small deviations from the exact center of the vortex caused by the interpolation onto a uniform grid cause the vortex particles to deviate from the exact potential flow solution and to “swirl” around the original vortex core location.

The next validation using this case is to simulate the diffusive effect of an Euler flow solution within the free-vortex code used above: a Laplacian smoothing operator was applied to the interpolated velocity field to approximate the effects of the vortex being diffused into the flowfield. Figure 4.8b shows that four passes of this smoothing operator applied to the interpolated velocity field resulted in smoother convection of the vortex particles, as expected. While the smoothing operator was only an approximation to the effect of trying to track a vortex in an Eulerian solver, it demonstrated the basic degeneracy that appears in the system: the vortex particles move outside the exact center of the vortex.
Figure 4.8: Vortex ring convection in free-vortex solver. Solid lines are the vortex filament location; symbols are the massless particle positions.
core and begin to swirl around at a small radius. The more diffuse the vortex, the less the particles swirl. Moreover, although they exhibit this “swirling” behavior in the face of a perfectly-preserved vortex, the particles do not show a tendency to diverge further from the vortex core, maintaining a relatively stable radial distance from the vortex core.

The next step was to implement the vortex particle technique in IBSEN and re-run the same case, allowing the Eulerian flow solver to convect the vorticity field and vortex particles to verify that this behavior held in the real system. Figure 4.9a shows an iso-surface of vorticity magnitude at three timesteps, clearly demonstrating the dissipative nature of the solver. By the third timestep shown the vortex is barely discernible in the flowfield, losing nearly all of its original strength as it convects. Figure 4.9b shows the particles at the same flow solution times. As expected, the particles stay quite close to the vortex core, exhibiting only relatively small swirl behavior. The more the vortex dissipates, the less of this behavior is expected and the smoother the massless particle convection with the mean flow.

4.6.2 2D Vortex-Airfoil Interaction

Next, a simple 2D vortex convecting past an airfoil at zero degrees angle of attack was run to numerically evaluate the amount of dissipation expected for this type of case. Fig 4.10 shows a schematic of the case. A Scully-model vortex was inserted into the flow 30 chords upstream of the airfoil quarter chord, 0.25 chords below the airfoil centerline, and a tracking particle was inserted at its core. Figure 4.11 shows the decay in circulation as the vortex and particle convect downstream, losing nearly 90% of the vortex’s initial strength before
(a) Iso-surfaces of vorticity magnitude.

(b) Massless particle positions.

Figure 4.9: Vortex ring convection in IBSEN.
interacting with the airfoil, despite tracking the location of the vortex as expected. During the interaction the circulation calculation is numerically unreliable due to the airfoil intersecting the integration circle, but after the interaction the vortex has weakened considerably further. This case demonstrates the behavior of the particle circulation calculation in the vicinity of a body (where the integration line may intersect the geometry), as well as providing a numerical illustration of the loss of vortex strength as the vortex convects in the Eulerian solver.

4.6.3 3D Vortex-Rotor Interaction

To verify that the system works as expected for three-dimensional flows, the next test case is based on an experiment conducted by Caradonna and Tung
Figure 4.11: Vortex circulation strength evaluated on a circle $3R_c$ from the vortex center.

(Ref. 135). In this case a finite-span NACA 0015 wing at 6 degrees angle of attack is placed upstream of a non-lifting rotor composed of two untwisted NACA 0012 rotor blades. The case is set up such that the wingtip vortex created by the upstream wing interacts with the downstream rotor at $\psi = 0^\circ$ and $\psi = 180^\circ$.

Figure 4.12 shows the overall case setup from a side view, and figure 4.13 shows a slice through the grid system used. In this simulation the vortex generator chord is 18 inches, and the rotor blade chord is 6 inches. The separation between the trailing edge of the vortex generator and the rotor blade tip is one rotor radius, 3.5 feet. This setup results in a vortex that is substantially larger than would be expected if the vortex were generated by the rotor itself, due to the much larger chord of the vortex generator. The vortex is also weaker than would be expected for a lifting rotor. These two factors combine to give a “best-
Figure 4.12: Side view of vortex interaction case setup showing the upstream vortex generator and small downstream rotor.

case scenario” for the CFD simulation of the vortex. The level 1 off-body grid is designed such that it has ten points across the rotor chord. This would ordinarily result in just two points covering the vortex core, one on each velocity peak (in the ideal case). However, because of the large size of this vortex this resolution actually results in four points across the core, a significant improvement in the resolution and capturing ability of the solver.

The vortex generator simulations were conducted using IBSEN on Penn State’s COCOA5 compute cluster. This cluster uses the AMD Opteron 6276 processor at a clock speed of 2.3 GHz. Each CPU consists of 16 cores, and each compute node has two CPUs. This run used one complete compute node, but only 24 cores. Scheduling was handled by the operating system. After grid splitting the case consisted of approximately 10.5 million grid points (a 9.02% increase over the unsplit grids). The interaction simulation ran for 32,400
timesteps and took a total of 140 hours (wall clock time), for an average of 15.5 seconds per timestep.

Figure 4.14a shows an isosurface of vorticity at a representative timestep of the case where the rotor blades are aligned at $\psi = 0^\circ$ and $\psi = 180^\circ$ azimuth (the interaction position). The rotor is placed such that the vortex passes just above the upper surface of the rotor. As seen in this image, the initial interaction of the vortex with the forward rotor blade disturbs the vortex and results in the vortex losing coherence by the time it encounters the aft blade. Figure 4.14b shows the same isosurface of vorticity but now includes the calculated path of the massless particles at this instant. The particles stay well-centered in the vortex, and clearly show the loss of coherence of the vortex as it interacts with
the rotor.

Figures 4.15a and 4.15b show the particle paths from two different views, colored by the local circulation. It can be seen from these plots that the local circulation steadily increases as the vortex convects downstream, until the point at which it encounters the rotor. This encounter disrupts the formation of the vortex and results in a rapid decrease in the local circulation.

Figure 4.16 shows a plot of the circulation versus downstream position. Included on the plot is the peak bound circulation on the vortex generator calculated by numerical integration of the Eulerian surface flow on the vortex generator at the time that the particle was released. This quantity is representative of the expected maximum circulation that the tip vortex should coalesce to in the ideal case. It can be seen that the bound circulation has been nearly reached by the developing vortex (as calculated from the IBSEN flowfield via numerical integration of a circle oriented perpendicular to the vortex path and located $10r_c$ from the particle position) within approximately 2.5 vortex generator chords downstream, which is exactly in-line with the expected rollup distance of 2-3 chords. However, at 2.33 chords downstream the vortex encounters the rotor, which disrupts its formation. The repeated blade passages can be seen in the oscillating values for circulation at the particles as they convect over the first blade, until the vortex is ultimately completely disrupted and the circulation falls off near the second blade position. This plot demonstrates that the Eulerian-predicted circulation at each particle position follows the expected roll-up at approximately the expected rate and strength. Due to the small domain size of this case and the closeness of the rotor/vortex interaction to the vortex generator, the formation of the vortex is prematurely disrupted and the
strength then falls off rapidly. Based on the results seen for the 2D case shown in Section 4.6.2 this rapid falloff is expected to occur even without the presence of the rotor once the vortex enters a region of the simulation where the grid is too coarse to maintain its form (in this case the rotor location and grid resolution reduction occur at approximately the same location so are indistinguishable from one another).

Finally, Figures 4.17a, 4.17b, and 4.17c show the vorticity at three positions during the formation of the tip vortex. These plots show that even while the calculated circulation is increasing, the peak vorticity (and accompanying peak velocity, etc.) is already decreasing in the core region. The ultimate result of this decrease is that, while the local circulation is well-calculated and preserved for a significant time (long enough to directly predict the roll-up process in this case as described above), the impulsive character of the vortex’s influence on the flowfield is rapidly lost. This supports the idea that the Eulerian flow solver can be used to directly predict the initial strength of a vortex, even when the local core behavior and vorticity are underpredicted. This predicted circulation strength can then later be used to reconstruct the magnitude of the BVI, and hence the generated noise. This procedure is demonstrated in the next section.
Figure 4.14: Isosurfaces of vorticity colored by local pressure.

(a) Isosurface of vorticity.

(b) Isosurface of vorticity including the trajectory of the massless particles.
Figure 4.15: Two views of the massless particle convection, colored by local circulation (calculated on a circle of radius $10r_c$).
Figure 4.16: Eulerian prediction of the local and bound circulation at the particle locations at a single instant in time, as a function of the particles’ distance from the rotor center.
Figure 4.17: Slices of vorticity magnitude at three different downstream positions during the formation of the tip vortex.
4.6.4 HART-II

The next test of the BVI noise prediction system is an analysis of a case from the HART-II rotor test (Ref. 136). The Higher Harmonic Control (HHC) Aeroacoustic Rotor Test (HART-II) was run in October 2001 at the German-Dutch Windtunnel (DNW) facility. For this case the HART-II rotor and fuselage geometry were used and two cases were run, one with the fuselage and one without. Figure 4.18 shows a schematic of the case setup with the HART-II fuselage geometry included. The elastic motion of the blade was simulated by matching the experimentally-measured tip motion (flap, pitch, and lag) but otherwise using a rigid blade. The rotor is a 40% dynamically and Mach-scaled BO-105 rotor, 4m in diameter, with 4 hingeless blades, chord 0.121m. The rotor precone is 2.5 degrees at the hub. The windtunnel model used a modified NACA 23012 rectangular blade with a trailing edge tab, square tips, and a -8 degree twist. The CFD geometry in this case omits the tab, using a simple NACA 23012 airfoil. The CFD case is set up to match HART-II baseline conditions with tunnel corrections. For this windtunnel test the rotor was run at 1041 RPM, $C_T = 0.0044$, a hover tip speed of 218 m/s, and an advance ratio of 0.15. This case is set up to match data point 128: test Part 1b, HART-I baseline, no HHC, with acoustic measurement taken. The wind tunnel model used a rotor shaft tilt of 5.4 degrees. The CFD runs are at a shaft tilt of 4.6 degrees to correct for installation effects (-0.8 degree correction). In addition, to better match the experimental rotor thrust the rotor collective pitch was lowered by one degree as compared to the experimental collective.

Figures 4.19 and 4.20 show representative slices of the HART II grid system used in this research.
Figure 4.18: HART case setup used in the with-fuselage simulations.
Figure 4.19: Top view of the HART-II overset grid system.

Figure 4.20: HART II body-fitted blade mesh slice.
All HART-II simulations were conducted using IBSEN on Penn State’s COCOA5 compute cluster. This cluster uses the AMD Opteron 6276 processor at a clock speed of 2.3 GHz. Each CPU consists of 16 cores, and each compute node has two CPUs. This run used one complete compute node, but only 16 cores. Scheduling was handled by the operating system. After grid splitting the case consisted of approximately 8.1 million grid points (a 3.76% increase over the unsplit grids). The interaction simulation ran for 18,000 timesteps and took a total of 44 hours (wall clock time), for an average of 8.81 seconds per timestep. The isolated-rotor simulation ran for 40 hours for an average of 7.92 seconds per timestep.

Figures 4.21a and 4.21b show top and side views of the tip vortex structures as predicted by the massless particle method. While the presence of the fuselage does have an impact on the particle positions downstream, in the more critical BVI locations the positions are very close. This is due to the slender shape of the HART II fuselage geometry.

Figure 4.22 shows four different disk loadings, one for each combination of with and without the fuselage, and with and without the particle reconstruction. The two baseline no-particles cases show similar loading patterns, though the presence of the fuselage is seen to impact the loading downstream of the hub. With the tracking particles included the BVI events become obvious. The inclusion of the fuselage has minimal impact on the advancing side BVI, but does slightly alter the interaction locations on the retreating side.

Figure 4.23 shows a comparison of the BVISPL directivity for variety of cases, including several other computation analyses from other research teams, using a wide variety of codes, taken from Ref. 137. (BVISPL is an integrated quantity
Figure 4.21: Comparison of tip vortex positions between the isolated (red circles) and interaction (blue squares) cases.

similar to OASPL, but calculated using only the 4–40 harmonics of the blade-passage frequency.) In that paper Lim presents the results from three different teams using Comprehensive Analysis suites to analyze the aerodynamics and aeroacoustics of the HART-II experiment. Each team used a different method, but all shared the basic framework of using a discrete vortex filament method to simulate the rotor aerodynamics.

Figure 4.24 shows the IBSEN results with and without the fuselage, but with no particle BVI reconstruction. Clearly the CFD alone is incapable of predicting the BVI noise. While the inclusion of the fuselage appears to improve the
prediction somewhat, it is still underpredicting by a substantial margin. This is expected for a CFD simulation of this fidelity, using low-order methods and a coarse grid. While more modern CFD techniques are available today, and more gridpoints can be used, CFD still underpredicts the BVISPL substantially. Because this research is focused on design-level simulations where runtime is critical and computational resources at a premium, this CFD simulation is representative of what level of BVI prediction can be expected from these simple, relatively inexpensive simulations.

Figure 4.25 shows the same results from IBSEN, but rescaled to show more detail (note that these plots are no longer using the same color scale as the experimental results and so cannot be used for direct comparison). It is clearer in these plots that the inclusion of the fuselage is having a relatively strong impact on the advancing side BVI, but is not significantly improving the retreating side BVI, which appears to be completely absent in this baseline simulation, despite the experimental evidence that it is in fact the dominant BVI noise event.

Figure 4.26 shows the results from IBSEN when the particle reconstruction method is enabled: the prediction is dramatically improved and corresponds well to both the other theoretical predictions as well as to the experimental results, resulting in a prediction that is slightly higher than the expected retreating-side peak, but lower than the predictions of Team B. Although the exact shape of the signal differs from both the experimental and other theoretical predictions, all of the predictions show considerable scatter in the noise footprint prediction. At least some of the difference can be explained by the use of a rigid-blade approximation rather than a fully-elastic simulation (as was used by the other theoretical methods shown). It is clear that the inclusion of
the particles provides significant improvement in the prediction of the BVISPL for this case.

The inclusion of the fuselage is more ambiguous. The retreating-side BVI is further strengthened when the immersed boundary fuselage model is included in the simulation, increasing the overprediction of the BVISPL in that region. However, the predicted BVISPL is still in line with the other theoretical predictions, and does show that the retreating-side BVI is expected to be somewhat higher than the advancing side. Given the nature of the HART-II model fuselage, which is quite compact and narrow, more significant changes to the acoustics were not expected when the fuselage was included.
Figure 4.22: Magnitude of the local acoustic loading vector over one rotor revolution for a representative blade in the HART-II simulation, with and without fuselage, and with and without vortex particle reconstruction of the BVI events. $\Psi = 0^\circ$ is at the top of the plots.
Figure 4.23: BVISPL in dB re:20\(\mu\)Pa (calculated for harmonics 4–40) for the baseline DP128 case, experimental and theoretical results, from Ref. 137. \(\Psi = 0^\circ\) is at the top of the plots.
Figure 4.24: BVISPL in dB re:20μPa (calculated for harmonics 4–40) for the baseline DP128 case, experimental and IBSEN results at the same scale, baseline CFD case with no BVI reconstruction. $\Psi = 0^\circ$ is at the top of the plots.
(a) IBSEN, no fuselage.  
(b) IBSEN, with fuselage.

Figure 4.25: BVISPL in dB re:20μPa (calculated for harmonics 4–40) for the baseline DP128 case, rescaled to show detail. Ψ = 0° is at the top of the plots.
Figure 4.26: BVISPL in dB re:20μPa (calculated for harmonics 4–40) for the baseline DP128 case, experimental and IBSEN results at the same scale, CFD case with added particle-based BVI reconstruction. Ψ = 0° is at the top of the plots.
4.6.5 Tiltrotor

As a final test case a representative modern heavy lift tiltrotor was analyzed. Due to the protected nature of the V-22 Osprey’s acoustic signature, several significant modifications were made to the case to ensure that the acoustic results presented here are not representative of the actual aircraft, but rather of a notional heavy-lift tiltrotor. In addition, all results are presented in normalized form, and only derived quantities are shown.

The rotor system used in this case is dual three-bladed counter-rotating rotor system created using a sequence of NACA four-digit airfoils, starting with the NACA 2409 at the tip and progressing through the 2412 and 2418 in the main blade section, with a short transition to a much thicker NACA 2428 at the inboard collar section of the blade. The blade is linearly twisted 30 degrees (far more than a standard helicopter due to the dual use of the rotor as a propeller in forward flight). The rotor RPM was set to 398.1, giving a hover tip Mach number of 0.707. Figures 4.27a, 4.27b, and 4.27c show the rotor blades used in this simulation. While designed to look like a reasonable tiltrotor blade, the airfoil sections used here are not those used in a real tiltrotor geometry, and no effort was made to ensure appropriate performance levels.

The geometry used in this case is a 3D CAD approximation of the V-22 fuselage obtained from http://www.3dcadbrower.com. The model consist of a triangular mesh in STL format with 53,517 polygons and 28,097 vertices, shown in 4.28. This model, intended for use in 3D game development, includes features such as deployed landing gear, pitot-static probes, antennae, nacelle actuation arms, and a model cockpit geometry including internal seats and control mechanisms, some of which can be seen in Figure 4.29. In addition, because this model
was not designed for flowfield analysis, there are a large number of degenerate triangles resulting in a computationally complex input geometry ideally suited to the new immersed boundary technique presented in Chapter 3. In any other type of flow solver this geometry would either be wholly inadequate, or would at least require significant cleanup and complexity reduction prior to its use in a flow simulation. Because the Inverse Distance-Weighted Average immersed boundary technique was developed specifically for use with coarse grids, any complex geometry features that are not adequately resolved by the grid are ef-
fectively smoothed over automatically by the solution procedure, without introducing the potential for numerically-divergent flow solutions.

Note that for computation time reasons, the engine nacelles were omitted from the calculation due to their intersection with the moving curvilinear blade meshes. While IBSEN is capable of performing the simulation with the nacelles intact, it was judged that the increased computational effort to handle the intersection of the moving blade meshes with the stationary nacelles was unnecessary, as the nacelles are quite thin and were not expected to have a strong impact on the positions of the rotor tip vortices at the BVI interaction locations.

Figure 4.30 shows a slice through the grid system at the center of the rotor system. A low-fidelity “collapsed-tip” body-fitted grid was used to represent the rotor blades, and a simple multi-level overset structured Cartesian mesh system was used to extend the grid system out to a sufficient distance so that the boundary conditions did not strongly influence the near-body flowfield. No attempt was made to include the fuselage in a finer level of detail mesh. An identical grid system was used in both the with- and without-fuselage cases.
The complete grid system consisted of the six curvilinear blade meshes and five levels of Cartesian mesh, for a total of approximately ten million gridpoints.

A flight condition was chosen to represent a strong BVI condition, as well as a condition in which it was anticipated that the fuselage would impact the positions of the rotor tip vortices. In this case the rotorcraft is in Mach 0.1 descending forward flight on a 6 degree glide path relative to the rotor tip path plane. Standard atmospheric conditions at sea level were assumed throughout. Four different cases are presented here: the simulation was run with and without the fuselage, and the acoustics for each case were calculated with and without the vortex interaction strength preservation particles method.

All tiltrotor simulations were conducted using IBSEN on Penn State’s COCOA5 compute cluster. This cluster uses the AMD Opteron 6276 processor
Figure 4.30: Rotor centerline slice showing the grid system used in the tiltrotor analysis.

at a clock speed of 2.3 GHz. Each CPU consists of 16 cores, and each compute node has two CPUs. This run used one complete compute node, but only 16 cores. Scheduling was handled by the operating system. After grid splitting the case consisted of approximately 9.5 million grid points (an 11.29% increase over the unsplit grids). The interaction simulation ran for 21,600 timesteps and took a total of 112 hours (wall clock time), for an average of 18.6 seconds per timestep. The isolated-rotor simulation ran for 106 hours for an average of 17.6 seconds per timestep. The conversion from an isolated rotor case to a case incorporating the fuselage geometry took a matter of hours due to the lack of geometry pre-processing required and the use of an immersed boundary technique.

Figures 4.31a and 4.31c show the momentum magnitude at a slice through
the middle of the rotor disks after seven rotor revolutions. The flowfield exhibits significant aperiodicity: one cause of aperiodicity is the deep stall the inboard blade sections are in due to their extremely high angle of attack (as compared to a conventional helicopter where the blades can be optimized purely for helicopter flight rather than both helicopter mode and airplane mode as in the case here). Another cause of aperiodicity is a small region of transonic flow near the advancing tips, where the flow separates off the NACA 2409 airfoil section due to the presence of a shock on the upper surface. These phenomenon combine to yield a highly unsteady flowfield: the influence of the fuselage is to confine more of this unsteadiness in the regions immediately adjacent to the rotors, significantly impacting their inflow.

Because IBSEN is an inviscid flow solver and is run at a low order of time accuracy it is not expected that the eddies seen here are accurate to the true flowfield around this aircraft, but their presence and their influence on the general characteristics of the tip vortex convection are expected to be representative.

Figures 4.32b and 4.32b show a snapshot of the positions of the rotor tip vortex filaments after seven revolutions. Note that this visualization does not take into account core-bursting. All filaments are shown regardless of whether they are burst or not. The influence of the chaotic nature of the flow in the separated regions is obvious in the positions of the tip vortices.

Figures 4.33a and 4.33b show a snapshot of the strengths of the rotor tip vortex filaments after seven revolutions (note again that this visualization does not take into account core-bursting). The absolute strengths of the vortices is relatively unaffected by the influence of the fuselage, with both cases showing very similar strengths: only minor differences are evident. Any difference in the
Figure 4.31: Snapshot of momentum magnitude on a slice through the rotor disks after seven rotor revolutions.
(a) Isolated rotors.  
(b) Rotors with fuselage.

Figure 4.32: Snapshot of tip vortex positions after seven rotor revolutions.

BVI noise between these two cases is expected to be caused by the difference in vortex positions, rather than a difference in their relative strengths.

Figure 4.34a shows the vortex filaments from both cases overlaid on one another. In this image it is possible to directly examine the difference between the two cases. Figure 4.34b shows the same thing, but zoomed in on just the rotors. Because of the substantial aperiodicity evident in this case it’s difficult to make any general statements about the vortex positions except to note that the inclusion of the fuselage model does affect their locations, and is therefore expected to affect their influence on the rotor noise.

All previous figures ignored core bursting. Figures 4.35a and 4.35b show a snapshot of the positions of the rotor tip vortex filaments over just the right-hand rotor after seven revolutions, with burst filaments removed from the visualization. Because of the nature of this particular case, much of the complex motion apparent in previous images was due to one particle in a filament passing
Figure 4.33: Snapshot of tip vortex strengths after seven rotor revolutions.

above the rotor blade and the next particle passing below it, where the filament itself is cut by the rotor blade. In that case the two particles diverge widely from one another. This does not influence the acoustics, however, as the burst model prevents these highly-stretched filaments from participating in the calculation.

Finally, Figures 4.36a, 4.36b, 4.36c, and 4.36d show the BVISPL for each of the four cases: with and without fuselage, and with and without particle-based BVI preservation. In the standard, non-preservation cases, the presence of the fuselage results in a small increase in the BVISPL in this case, particularly along the fuselage centerline. Including the particle preservation method adds approximately 3-4 dB to the overall BVISPL prediction, and further enhances the difference between the isolated rotor and fuselage interaction cases. The asymmetry shown in the results is due to the use of only two rotor revolutions of acoustic data in the BVISPL calculation. Because of the long-timescale aperiodicity in this case, the results are only capturing a short segment of the BVI events
Figure 4.34: Overlay of tip vortex positions after seven rotor revolutions. Red line is interaction case, blue is isolated.
(a) Isolated rotors.  
(b) Rotors with fuselage.

Figure 4.35: Snapshot of tip vortex positions after seven rotor revolutions, with burst filaments removed.

on one side of the rotor. Over time, these events are expected to shift. Nevertheless this simulation demonstrates the capability of the flow solver to include an extremely complex fuselage model with minimal user effort, and the potential for the inclusion of that model to influence the BVI noise, particularly when coupled with the new vortex interaction strength preservation method developed in this research. Note that this does not account for the acoustic scattering influence of the fuselage. The acoustic differences shown here are purely due to the fuselage’s direct impact on the rotor loading.
(a) Isolated rotors, no BVI preservation.

(b) Rotors with fuselage, no BVI preservation.

(c) Isolated rotors with BVI preservation.

(d) Rotors with fuselage with BVI preservation.

Figure 4.36: BVISPL comparison for the tiltrotor case.
Conclusions and Future Work

In this research, three principal contributions were made:

- A new Immersed Boundary technique was developed that is stable in the face of geometric degeneracies and coarse grid, and can be used on arbitrarily coarse grids without concern for the resolution of the underlying input body geometry.

- A new blade-vortex interaction (BVI) strength preservation technique was developed that takes advantage of Eulerian flow solvers’ ability to predict complex interactional aerodynamic phenomena while mitigating their inability to capture the expected strength of the BVI event in the acoustics calculations.

- A new flow solver was developed incorporating both of these techniques and emphasizing simple setup and integration with acoustics simulations for improved turnaround time during rotorcraft design.

The new Inverse Distance-Weighted Average (IDWA) Immersed Boundary technique developed in this research was designed to allow the easy incorpo-
ration of a rotorcraft fuselage in a CFD simulation originally designed as an isolated rotor case. It enables this inclusion without significant human or computational time required. The technique uses a low-fidelity (but highly stable) reconstruction of the flowfield at the immersed surface, which is then extrapolated to the first layer of grid points outside the boundary. In addition, numerous techniques are used to add robustness to the geometry handling and to resist the impact of degeneracies on the flow solution. This technique was demonstrated in both 2D and 3D, for both simple and extremely complex geometries. It is particularly valuable for acoustics simulations where only the forces acting on the rotor blades themselves are required, and a low-fidelity representation of the geometry is acceptable as long as it provides approximately the correct influence on the rotor inflow.

The new BVI technique uses a novel implementation of a combined Eulerian-Lagrangian simulation in which a conventional Eulerian flow solver is used to calculate the formation stages of a rotor tip vortex (including its circulation strength), but is then used to convect Lagrangian particles through the flowfield to better capture the impulsive nature of BVI noise as an acoustics post-processing step. During the creation of the acoustic data surface loading information, the new technique is used to reconstruct an approximate value for the compact blade loading that takes into account the presence of a physically-realistic discrete vortex filament. This technique was shown to dramatically improve the BVISPL predictions for the HART-II case as compared to the baseline low-resolution Euler simulation, and similarly impacted a hypothetical tiltrotor case.
In the following sections further analysis of and extensions to the research presented here are proposed.

5.1 Surface Treatment Techniques

A number of immersed boundary surface treatment techniques were examined, ranging from the simplest classical formulation up to highly sophisticated surface reconstruction techniques. It was found that most of the classical techniques rely on the solution of the viscous Navier-Stokes equations and their attendant grid resolution requirements for their effectiveness, and most of the penalization techniques yield inadequate results when used on coarse grids with inviscid flow equations. The more complex surface reconstruction techniques fare somewhat better, but suffer from stability problems when coupled with highly-complex geometries and high-gradient flowfields in under-resolved regions of the grid. Since one of the driving considerations for this flow solver is the use of coarse grids and complex geometries, this poses a serious concern. To address it, a much simpler, more approximate method was developed for applying the immersed boundary condition. This new technique is stable even in regions where the user has under-resolved the flowfield (perhaps because they are not interested in the locally-complex flow). The new technique shows excellent agreement with OVERFLOW for simple thin airfoil cases, and qualitatively good results for more complex cases.

There are several further research paths that may be able to yield improved accuracy and/or improved performance over the baseline IDWA technique. First, it may be possible to use the surface curvature information already present
in the computation to dynamically determine which immersed boundary technique to use based on the radius of curvature and local grid spacing. The method could potentially automatically switch between a more sophisticated immersed boundary technique and the lower-fidelity IDWA technique as the geometry requires. In a second avenue for further study, the IDWA method itself could potentially be improved, particularly for transonic cases, by identifying shocks prior to the averaging process in order to ensure more physically consistent behavior, rather than the current default of blindly averaging points on either side of the shock. Third, one remaining problem area for the current technique is in long, narrow passages with few gridpoints: regions with just enough points that computation is required in the region, but not enough that the IDWA algorithm has a large number of neighboring points to average from. While these passages do not often occur in helicopter cases and did not affect the present research, it remains a challenge to find a technique that is stable in such areas.

5.2 Geometry Handling

The current approach implements relatively robust geometry handling, both in the actual geometry input code as well as in the new techniques presented for mitigating the effects of geometric degeneracies on the final flow solution.

From a speed consideration, it may be worthwhile to take advantage of some of the sophisticated surface mesh coarsening features of the already-included code libraries. The search for intersecting geometries must be conducted at every timestep in regions where grids and bodies are not moving in unison. This
search is $O(\log n)$ where $n$ is the number of triangles. If large reductions in the number of triangles can be achieved via coarsening, some speed improvement could be realized. This is especially useful at the case setup phase, where the user might wish to run the code for a single timestep dozens of times to perfect the grid system: this type of optimization is unlikely to have any major impact on the overall runtime of the flow solver for most cases.

5.3 Blade-Vortex Interaction Capturing

Further research is required to fully assess the capabilities of the BVI-capturing system implemented here, and to examine its efficacy for capturing other types of wake-structure interactions. While basic analysis was presented demonstrating that the particles correctly convect along with the flow, are capable of calculating and retaining a parameter related to expected vortex strength, and that this procedure can be used to reproduce reasonably accurate BVI noise in the HART II simulation, further verification of the technique is desirable. The major obstacle to this is lack of available BVI acoustic data coupled with adequate experimental information about the vortex itself. The HART-II case represents the highest-fidelity BVI analysis to date. Future wind tunnel studies could provide larger PIV fields of view, which would help determine whether the choice of vortex model was adequate, as well as simply providing additional data points for comparison.

Improvements to the technique could be achieved in several ways: first, it may be possible to replace the assumed $2\pi$ lift curve slope with a slope dynamically determined by the CFD as the case runs. This would continue to allow
the method to avoid 2D airfoil lookup tables as well as potentially taking into account any expected lift hysteresis (though the nature of the calculation would still be quite approximate). Next, the core-bursting model used in this research is very rudimentary, and could be improved by more careful modeling of the actual physical behavior of a vortex as it comes into contact with a solid surface. The core growth model may also be improved by further examination of experimental results and/or additional analysis of the HART-II data to more accurately assess the initial conditions. Finally, the entire technique presently used many short, straight vortex segments. It may be advantageous to use fewer, curved segments to represent the tip vortex path.

5.4 Code Speed Optimization

This code has been designed for nearfield acoustics calculations of large-scale highly-complex heavy lift rotorcraft. While preliminary code optimizations have been completed, more sophisticated treatment of the available computational power, especially on multi-core systems, could provide substantial speed improvements. While not a focus of this research, some preliminary analysis has been undertaken to identify possible regions for improvement.

The serial performance of the code is approximately 25% slower than an OVERFLOW run at the same order of accuracy. This seems to be due primarily to OVERFLOW’s more sophisticated implementation of the linear solver as well as much more aggressive overall optimization related to the in-memory structure of the code. While some of this speed difference could be regained through optimization, OVERFLOW is an exceptionally mature and well-optimized code,
so it is likely that IBSEN will always lag in that respect.

In parallel, however, overall it is seen that IBSEN’s scalability is relatively poor, and that the bulk of the problem appears to be the actual data transmission (rather than any oversetting or interpolation computations). More efficient transmission algorithms may help to alleviate this problem. In addition, on shared-memory nodes it is likely that using OpenMP to perform fine-grained parallelism would be substantially more efficient than using MPI within the node. OVERFLOW, for example, implements a message-passing scheme for communicating between physical compute nodes, but uses OpenMP to perform shared-memory parallel computation within the multi-core node itself.

Finally, most optimization work has focused on the flow solution procedure, with little effort made to improve the immersed boundary interior/exterior and grid edge intersection algorithms. All of the cases presented in this dissertation used geometries that never moved relative to the grids they were embedded in. IBSEN supports this type of motion, but enabling it greatly slows down the calculations. There are many types of optimizations that could be made to minimize the impact of this type of scenario, however, and speed improvements will greatly increase the utility of the system for those types of cases.

5.5 Conclusions

A new computational approach has been developed for acoustics calculations of highly complex interactional flows in large heavy-lift rotorcraft. This code has been specifically designed for the computational aeroacoustics problem, while still allowing relatively short runtimes for full-configuration computations. It
supports highly complex bodies and body motions, automatic off-body grid generation, and simple coupling with the PSU-WOPWOP acoustic propagation code. A new technique for handling the immersed boundary condition has been developed to facilitate highly-stable runs with complex geometries and very coarse grids, and a new technique for approximating the effects of blade-vortex interactions on the acoustics of the system has been implemented. These contributions significantly improve our ability to compute and analyze the acoustics of complex heavy-lift rotorcraft and provide a strong foundation for future research in this area.
The Immersed Boundary Solver for Environment Noise (IBSEN) was designed to be (relatively) easy to use, especially for very complex geometries. Its handling of body motion is exactly analogous to PSU-WOPWOP’s, so familiarity with those inputs is useful—though not required—for using IBSEN. The most complicated part of setting up an IBSEN run is designing the grid system and debugging the body motions, both of which are primarily a manual process. This User’s Manual will guide you through the step necessary to set up an IBSEN case.

A.1 Compiling and Installing IBSEN

IBSEN is distributed as C++ source code: before using it, it must be compiled. For detailed instructions see the “README.installation” file distributed with the source code. In general the steps are:

1. Compile and install the necessary libraries

2. Create the configuration files
3. Run the configure script

4. Compile the code

5. Install the executable and support files

IBSEN depends on a number of libraries, many of which are standard system installs, especially on clusters. When possible, it is best to request that these libraries, if absent, be installed by the system administrator, rather than installing them in your home directory. IBSEN will work either way, however. The “standard” libraries required are:

- GMP — The GNU Multiple Precision Arithmetic Library.
- MPFR — A C library for multiple-precision floating-point computations with correct rounding.
- Xerces-C — A validating XML parser written in a portable subset of C++ by the Apache project.
- BLAS — Routines that provide standard building blocks for performing basic vector and matrix operations.
- Boost — (Only needed for older compilers) Support for shared pointers to reduce the possibility of memory leaks.

Note that high-end BLAS and LAPACK libraries optimized for your machine are not required, though they will provide a small speed increase. Boost is only
required for compilers that do not natively support the `std::shared_ptr` smart pointer defined in the C++ Technical Report 1 (Ref. 138), for example the GCC 3.X.X series.

Non-standard libraries required are:

- **blitz++** — A C++ class library for scientific computing which provides performance on par with Fortran 77/90 for certain types of large mathematical operations. (formerly available at oonumerics.org/blitz, now defunct.)

- **tvmet** — A Tiny Vector and Matrix template library uses Meta Templates (MT) and Expression Templates (ET) to evaluate results at compile time—which makes it fast for low order (tiny) systems. (tvmet.sourceforge.net)

- **Loki** — A C++ library of designs, containing flexible implementations of common design patterns and idioms. (loki-lib.sourceforge.net)

- **CGAL** — A collaborative effort to develop a robust, easy to use, and efficient C++ software library of geometric data structures and algorithms. (www.cgal.org)

All of these libraries are included with the source distribution of IBSEN, and should be installed following the instructions in the “README.installation” file provided with the distribution. Once the libraries are installed you are ready to compile IBSEN. In some cases more recent versions of the libraries can be used if desired.

If you checked IBSEN out directly from the Subversion repository, the first step is to create the configuration scripts by running:

```
prompt$> ./autogen.sh
```
This will create a number of files that are not normally committed to the Subversion repository. Once this is complete, or if you have received a source distribution from Penn State, run:

```bash
prompt$> ./configure
```

If you wish to change any of the default options, you can see a list of available variables by running:

```bash
prompt$> ./configure --help
```

In particular, you may wish to change the standard installation location, either to install in the current directory, or into your home directory. For example:

```bash
prompt$> ./configure --prefix=$PWD
```

will install IBSEN and its related files into the current directory (in subdirectories “bin” for the executable and “share” for the support files). Once the configuration script has completed, compile the executable:

```bash
prompt$> make
```

And install it in the selected location:

```bash
prompt$> make install
```

This last command will copy the executable, ibsen, and the input definitions file, CaseDefinition.xsd into the appropriate locations on your system. The default installation locations are system-dependent, but generally `/usr/local/[bin|share]`, or `$PREFIX/[bin|share]` if you set a different installation prefix when you ran the configuration script.
A.2 File Overview and Basic Setup

Running IBSEN in general requires a minimum of three files:

- The ibsen executable itself, generally installed in $PREFIX/bin (where prefix defaults to /usr/local but may be set when running the configure script). The path to this executable must be in your PATH environment variable, or you must call it using the full path.

- The CaseDefinition.xsd file, generally installed in $PREFIX/share (where prefix defaults to /usr/local but may be set when running the configure script).

- Your case input deck, an XML file you create to define the case. The bulk of this documentation is geared towards helping you create that file.

In addition your input deck may reference any number of additional files, for example geometry inputs, motion files, etc.

A.3 CaseDefinition.xsd

The “CaseDefinition.xsd” file describes the format of the input file to IBSEN, allowing it to easily check your input file for syntax errors, and (hopefully) write out a descriptive error message. This file can be opened in a text editor and exactly describes the required inputs to IBSEN, if you know how to read it! This manual will describe the required inputs in relatively plain English, as well as show you the corresponding lines in the XSD file (if you’re curious, “XSD” stands for “XML Schema Definition”). This file is usually installed in a standard
location, but since your input deck must state the location of this XSD file, you can place it anywhere you like.

### A.4 Main input file

Your main input file is an XML file, which is roughly analogous to a Fortran Namelist file, if you are familiar with those. Its formatting is different, but it serves the same purpose: it is a list of variables and their values. It is designed to be easy to be read and written by a computer, as well as by a person. As you will see, utilities exist that will write out sections of this file for you, if desired.

To get started on your first case this manual shows you how to create the XML input file from scratch. Of course, most of the time you will simply copy an old input file into your new case directory and modify it to fit your new case. Start by creating a new input file: it is good practice to end it with the “.xml” suffix, since many text editors will recognize this and perform syntax highlighting for you:

```
prompt$ > touch NACA0009_03degrees.xml
```

#### A.4.1 Header

The header of the XML input file always looks like this:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<case xmlns="http://www.psu.edu"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.psu.edu /usr/local/share/CaseDefinition.xsd">
```
This initializes IBSEN with some information about the XSD file that this XML goes with: the XSD file is `CaseDefinition.xsd`, and it is from the `www.psu.edu` domain. In general you can ignore this part of the file, but you must include it, and it must look exactly like this (formatting does not matter, but spelling does). If you installed the CaseDefinition file on your system someplace (perhaps by using "make install") it will be placed in your system's data directory. On Linux this defaults to `/usr/local/share/`, but your system may be set up differently (or you may have specified a different prefix). In that case, replace "`/usr/local/share/CaseDefinition.xsd`" with the actual location the the XSD file, for example "`/home/chennes/IBSEN/share/CaseDefinition.xsd`". If you do not wish to refer to a system-wide XSD file, just copy the file into your run directory and leave the full path off the schema location in your input file, just using "`CaseDefinition.xsd`”—IBSEN will then look for the scheme file in the current working directory when you run the case.

Following this header we begin the actual definition of our case:

```xml
<case>
  <name>NACA 0009 Test Case</name>
  <description>2D NACA 0009 at 3 degrees AOA, Mach 0.2 flow</description>
  <!-- <restartfile>TestRestart.q</restartfile> -->
  <restartstep>50000</restartstep>
  <nsteps>100</nsteps>
  <dt>0.00075</dt>
  <adaptivetimestep>true</adaptivetimestep>
  <basename>NACA0009_2D_03_degrees</basename>
</case>
```

As you can see, the format of the variable definition statements is:
This will get a little more complicated later in the file. In the XSD file defining the input format, there are several additional variables not listed in the example. In the XSD file, the start of the “<case>” element is defined like this:

```
<case
  <xs:element name="case"/>

The start of the case element.
```

Below that are the definitions of all the variables a case element supports: the example above lists some of the available variables. Some of the variables are required: in the XSD file this is specified by the “minOccurs='1'” attribute, while some are optional, with “minOccurs='0'.” All of the available case variables are listed below, along with the line defining them in the XSD file and a brief description. The remainder of this section of the documentation describes all the possible inputs to IBSEN, all of which are sub-elements of this main <case> element.

A.4.2  <case> Element Inputs

The following variables are all of the current possible inputs to the main <case> element, along with their specifications from the XSD file in small text, and a brief description of the variable. Many of these variables are optional an may be omitted: in the XSD file this is specified by the “minOccurs='1'” attribute, while some are optional, with “minOccurs='0'.” The data type of the input is specified in the XSD file as well, e.g. “xs:string,” “xs:boolean,” “vector,” etc. In most cases the input is what you would expect for a given data type. Exceptions to that are non-standard types (i.e. those that do not begin with “xs:’’): these are type actually defined in the XSD file. Where necessary the documentation below notes these along with a description or example of the input type.
name
<xs:element name="name" type="xs:string" minOccurs="1" maxOccurs="1"/>
A short, one-line name for the case, used in debugging outputs.

description
<xs:element name="description" type="xs:string" minOccurs="0" maxOccurs="1"/>
A longer description of the case, printed out at the beginning of the run. As a best practice, put whatever case information you think is pertinent here: if you store the output from the case all together this should be an extensive enough description to allow future replication or at least understanding of the case.

debugging
<xs:element name="debugging" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
Set to “true” to get additional output while the case runs. Not generally useful unless the case is hanging.

dryrun
<xs:element name="dryrun" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
Set to “true” to disable the actual fluid dynamic calculation, therefore only moving the bodies, cutting overset holes, calculating grid/body intersections, and writing out grids and body files as necessary.

disableinteriorexterior
<xs:element name="disableinteriorexterior" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
If (and only if) this case is a dry run as specified above, set to “true” to disable the body interior/exterior detection and just perform the overset hole-cutting.

forcesplittings
<xs:element name="forcesplittings" type="xs:nonNegativeInteger" minOccurs="0" maxOccurs="1"/>
For debugging purposes, set to a positive integer to force the code to split the grids at least this number of times. Not generally useful except to examine the potential splittings when running on only a single processor.
**maxprocessespernode**

```xml
<x:element name="maxprocessespernode" type="xs:nonNegativeInteger" minOccurs="0" maxOccurs="1"/>
```

To work around clusters that don’t nicely deal with multi-core architectures you can ask the queuing system to give you all the cores and then use this value to limit the actual run to only a subset of them. For example, on an eight-core CPU you might ask the queuing system for all eight, but then set this variable to six, to avoid memory bandwidth problems. The use of this variable will be highly cluster- and CPU-dependent. The internal variable defaults to infinity if this is not set.

**numopenmpthreads**

```xml
<x:element name="numopenmpthreads" type="xs:nonNegativeInteger" minOccurs="0" maxOccurs="1"/>
```

If OpenMP is enabled, and the environment variable `OMP_DYNAMIC` is not used to disable dynamic allocation of threads, this variable sets the number of threads allocated to OpenMP. In general, set to the number of available compute cores on the node. Best used in conjunction with requesting all processors on a node using your queuing system, setting `maxprocessespernode` to 1, and setting this variable to the number of compute cores. Note that the current OpenMP implementation is not well-tested and the implementation need substantial work before it should be used in simulations.

**useiblankingstandard**

```xml
<x:element name="useiblankingstandard" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
```

If set to true, Plot3D restart grid files are written such that the iBlanking conforms to the iBlanking standard. This is useful when using FieldView to visualize the data, as FieldView does not allow customization of the blanking used in visualization.

**restartfile**

```xml
<x:element name="restartfile" type="xs:string" minOccurs="0" maxOccurs="1"/>
```

The filename of the solution file to restart from. This file must match the grid setup specified in the rest of the main input deck.
**restartstep**

```xml
<xs:element name="restartstep" type="xs:nonNegativeInteger" minOccurs="0" maxOccurs="1"/>
```

The timestep that is stored in the specified restart file. Used to ensure the correct output filenames are used in the restart.

**restartxmlfile**

```xml
<xs:element name="restartxmlfile" type="filename" minOccurs="0" maxOccurs="1"/>
```

If an XML file is provided for the restart, it is allowable to restart from a non-matching set of grids: the solution files are read in based on the grid specification in the restart XML file and the solution is interpolated onto the grids specified in the current case’s XML file. This is a memory-intensive operation, so do **not** specify a restart XML file if restarting with the same grids.

**dt**

```xml
<xs:element name="dt" type="xs:double" minOccurs="1" maxOccurs="1"/>
```

For a non-adaptive timestepping case, the physical timestep to take. For an adaptive case, the starting timestep, recalculated before the run, but must be positive and non-zero.

**adaptivetimestep**

```xml
<xs:element name="adaptivetimestep" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
```

Set to “true” to adjust the timestep size at each timestep so that the maximum CFL in the field is $cfl_{max}$ (see below).

**cflmax**

```xml
<xs:element name="cflmax" type="xs:double" minOccurs="0" maxOccurs="1" default="0.5"/>
```

In conjunction with adaptivetimestep, this sets the CFL that the case is limited to.

**nsteps**

```xml
<xs:element name="nsteps" type="xs:nonNegativeInteger" minOccurs="1" maxOccurs="1"/>
```

The number of timesteps to take. May be zero, in which case all setup and initial condition calculation and output is performed prior to exiting the run. If this case is a restart, this number specifies the maximum number of the timestep to stop at. That is, if you are restarting at timestep 5 and nsteps is 10, five more steps will be run (stopping after timestep 10).
### basename

<xs:element name="basename" type="xs:string" minOccurs="1" maxOccurs="1"/>

The prefix of all of the output files generated during the run.

### outputdirectory

<xs:element name="outputdirectory" type="xs:string" minOccurs="0" maxOccurs="1"/>

The directory in which to place the output files. Defaults to the current working directory.

### tempdirectory

<xs:element name="tempdirectory" type="xs:string" minOccurs="0" maxOccurs="1"/>

A temp directory to use. May be local to a cluster node (i.e. non-NFS) if desired.

### restartwritecontroller

<xs:element name="restartwritecontroller" type="intermittentcontroller_type" minOccurs="0" maxOccurs="1"/>

Controls when Plot3D “restart” files are written. See documentation of the “intermittentcontroller” type for details, Section A.4.3

### timeaveragedsolutionwritecontroller

<xs:element name="timeaveragedsolutionwritecontroller" type="intermittentcontroller_type" minOccurs="0" maxOccurs="1"/>

Controls when time-averaged solution files are written. See documentation of the “intermittentcontroller” type for details, Section A.4.3

### distributedforcewritecontroller

<xs:element name="distributedforcewritecontroller" type="intermittentcontroller_type" minOccurs="0" maxOccurs="1"/>

Controls when surface force files are written. See documentation of the “intermittentcontroller” type for details, Section A.4.3

### integratedforcewritecontroller

<xs:element name="integratedforcewritecontroller" type="intermittentcontroller_type" minOccurs="0" maxOccurs="1"/>

Controls when integrated force data is written. See documentation of the “intermittentcontroller” type for details, Section A.4.3
**visualizationwritecontroller**

```xml
<xs:element name="visualizationwritecontroller" type="intermittentcontroller_type"
minOccurs="0" maxOccurs="1"/>
```

Controls when visualization files are written. These files are in one of the VTK file formats, suitable for viewing in ParaView, or any other software that supports that format. They are typically about 25 percent smaller than the equivalent restart file, and in some cases can be used to visualize data that cannot be written in Plot3D format. See documentation of the "intermittentcontroller" type for details, Section A.4.3.

**progressdatauploader**

```xml
<xs:element name="progressdatauploader" type="progressdatauploader_type"
minOccurs="0" maxOccurs="unbounded"/>
```

If IBSEN was compiled with CURL support, it can push progress data out to an external server (if the cluster it is running on supports access to the requested resource from the nodes the code is running on). See the documentation on progressdatauploader below for details, Section A.4.3.

**density**

```xml
<xs:element name="density" type="densityWithUnits" minOccurs="1" maxOccurs="1"/>
```

Freestream density. Units may be specified as an attribute, e.g. `<density units="kg/m\^3">1.225</density>`. Possible units are “slugs/ft^3”, “kg/m^3”, and “nondimensional” (in which case density should be one). Defaults to “nondimensional”.

**pressure**

```xml
<xs:element name="pressure" type="pressureWithUnits" minOccurs="1" maxOccurs="1"/>
```

Freestream pressure. Units may be specified as an attribute, e.g. `<pressure units="Pa">101325</pressure>`. Possible units are “Pa”, “psi”, “lb/ft^2”, and “nondimensional” (in which case it should be nondimensionalized by freestream density and speed of sound). Defaults to “nondimensional”.

**temperature**

```xml
<xs:element name="temperature" type="temperatureWithUnits" minOccurs="1" maxOccurs="1"/>
```

Freestream temperature. Units may be specified as an attribute, e.g. `<temperature units="K">287.15</temperature>`. Possible units are “K”, “R”, “C”, “F”, and “nondimensional” (in which case the temperature should be one). Defaults to “nondimensional”.
**gamma**

```xml
<xs:element name="gamma" type="xs:double" minOccurs="1" maxOccurs="1"/>
```
Specific heat ratio.

**reflength**

```xml
<xs:element name="reflength" type="lengthWithUnits" minOccurs="0" maxOccurs="1"/>
```
The reference length is used for internal nondimensionalization and defaults to 1 if not set. This is usually adequate, as in the present inviscid code nothing is nondimensionalized with respect to length.

**re**

```xml
<xs:element name="re" type="xs:double" minOccurs="0" maxOccurs="0"/>
```
The Reynolds number. Not used in the present code.

**freestreamvelocity**

```xml
<xs:element name="freestreamvelocity" type="velocityWithUnits" minOccurs="0" maxOccurs="1"/>
```
Freestream velocity. Defaults to zero in all directions. “velocityWithUnits” is an internally-defined type in the XSD file and has three subcomponents, one for each cardinal direction. For example, 

```xml
<freestreamvelocity units="m/s"> <x>330.7</x> </freestreamvelocity>
```
Omitted components default to zero. Possible units are “m/s”, “ft/s”, and “Mach number”. Defaults to “Mach number” if not specified.

**cutterstouseforparticles**

```xml
<xs:element name="cutterstouseforparticles" type="string_vector" minOccurs="0" maxOccurs="1"/>
```
The particles get pushed to the exterior of any cutters listed here, as though they were solid bodies. In a normal run these cutters should be extracted from the grids representing the surface of any physical objects that have curvilinear meshes representing their surfaces. This will typically double the number of cutters used to represent a surface in a curvilinear mesh.

**objecttree**

```xml
<xs:element name="objecttree" type="objecttree_type" minOccurs="0" maxOccurs="1"/>
```
The main <objecttree> object. See further documentation in Section A.4.4.
Group: Solvers

Following the object tree is a list of solvers, automatic grid generators, acoustic data surfaces, and/or massless particle release points. These may be specified in any order, and there may be any number of them. See documentation of the individual data types in the sections below.

A.4.3 Intermittent Controller Inputs

Many sections of IBSEN are not executed every timestep: for example, restart files are typically written at a much lower frequency. Massless particles may be released frequently, but not started until later in the run, and flowfield averaging may be started and stopped at particular timesteps to capture a single periodic event, etc. All of these are controlled by IntermittentController objects: these simple objects are created directly by the user in the XML input file and can take two forms, the “simple” form and the “advanced” form. At its most basic, an IntermittentController specifies a starting timestep, a stopping timestep, and a timestep increment. The starting and stopping steps default to zero and infinity, respectively, so in practice many times only “increment” must be specified. The simplest controller might look like this (using <restartwritecontroller> as an example):

\[
<\text{restartwritecontroller}><\text{increment}>1000</\text{increment}></\text{restartwritecontroller}>
\]

A more complicated controller might be:

\[
<\text{timeaveragedsolutionwritecontroller}>
<\text{start}>1000</\text{start}>
<\text{stop}>2000</\text{stop}>
<\text{increment}>10</\text{increment}>
</\text{timeaveragedsolutionwritecontroller}>
\]
Finally, an arbitrary number of ranges may be specified: if any ranges overlap, the controller activates at all matching timesteps:

```
<timeaveragedsolutionwritecontroller>
  <range>
    <start>1000</start>
    <stop>2000</stop>
    <increment>10</increment>
  </range>
  <range>
    <start>2000</start>
    <stop>4000</stop>
    <increment>20</increment>
  </range>
</timeaveragedsolutionwritecontroller>
```

Note that in this case timestep 2000 matches both ranges: the controller is therefore activated at timestep 2000, but this does not cause the controller to be activated twice. A controller can only either be active or inactive at a given timestep.

---

**start**

```xml
<xs:element name="start" type="xs:nonNegativeInteger" minOccurs="0" maxOccurs="1"/>
```

The first active timestep. Do not use if using `<range>`.

---

**stop**

```xml
<xs:element name="stop" type="xs:nonNegativeInteger" minOccurs="0" maxOccurs="1"/>
```

The last active timestep. Do not use if using `<range>`.

---

**increment**

```xml
<xs:element name="increment" type="xs:nonNegativeInteger" minOccurs="0" maxOccurs="1"/>
```

The increment between active timesteps. Do not use if using `<range>`.
If multiple ranges are desired, a sequence of <range> objects is used. Each range object has a start, stop, and increment variable as described above. If a range object is specified, the start, stop and increment values for the top-level controller tag are ignored.

A.4.4 Immersed Objects

In the <case> element, after the variables for the case itself are specified, there is an optional specification for an <objecttree>: this is where any physical geometries are input that the fluid must flow around (i.e. these are the immersed boundaries). As is indicated by the name, this is a tree structure that allows the specification of complex hierarchies of objects (and object motions) by splitting it up into its component parts. For example, in a typical helicopter case, the following hierarchy might be used:

(Image of helicopter hierarchy here)

In XML this would be defined like

```xml
<objecttree>
  <name>Outer helicopter group</name>
  <plot3dobject><name>Fuselage</name></plot3dobject>
  <objecttree>
    <name>Main rotor</name>
    <objecttree>
      <name>Rotor blade 1 group</name>
      <plot3dobject><name>Rotor blade 1</name></plot3dobject>
      <periodicrotation><name>Flapping rotor blade 1</name></periodicrotation>
    </objecttree>
    <objecttree>
      <name>Rotor blade 2 group</name>
      <plot3dobject><name>Rotor blade 2</name></plot3dobject>
      <periodicrotation><name>Flapping rotor blade 2</name></periodicrotation>
    </objecttree>
    <aperiodicrotation><name>Rotor RPM</name></aperiodicrotation>
  </objecttree>
</objecttree>
```
An objecttree XML entry has the following structure:

<table>
<thead>
<tr>
<th>XML Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>A name for this level of the tree. Must be unique</td>
</tr>
<tr>
<td>sphere</td>
<td>A sphere object: see Section A.4.4.1</td>
</tr>
<tr>
<td>box</td>
<td>A box (e.g. a cube with potentially unequal sides): see Section A.4.4.2</td>
</tr>
<tr>
<td>cylinder</td>
<td>A circular cylinder: see Section A.4.4.3</td>
</tr>
<tr>
<td>stlobj</td>
<td>An object input from an STL (STereoLithography) file, typically exported</td>
</tr>
<tr>
<td>plot3dobj</td>
<td>An object input from a Plot3D file (e.g. a surface grid). See Section A.4.4.5</td>
</tr>
</tbody>
</table>

(Many variables omitted in this example: this shows the structure of the input, not the actual XML as it would be written in the input deck).

```xml
<aperiodicrotation>
  <name>Fuselage angle of attack</name>
</aperiodicrotation>
</objecttree>
```
A sub-tree: object trees may contain any number of sub-trees.

Group: Motions

Any number of motions. See Section A.4.5.

A.4.4.1  <sphere>

Sphere is one of several analytically-defined objects useful for testing or rudimentary object construction. It has the advantage over inputting a surface grid of being much faster for interior/exterior and intersection calculations, and the disadvantage of being unable to output surface pressures (because there is no mesh to write them on). The sphere is created centered at the origin, but a translation motion may be applied to move it to any location.

name

A name for the sphere

radius

The radius of the sphere.

iscutter

Whether this object is a cutter (set to true) or a normal object (set to false).

Group: Motions

Local motions for the sphere. See Section A.4.5.
A.4.4.2  <box>

Like Sphere, Box is an analytically-defined object used for basic testing and object composition. It is also useful as an overset grid cutter in cases where a closely-conforming box is adequate to blank out the interior points in a curvilinear mesh that oversets other meshes (most commonly used as the cutter for a rotor blade).

<table>
<thead>
<tr>
<th>name</th>
<th>&lt;xs:element name=&quot;name&quot; type=&quot;xs:string&quot;/&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>A name for the box.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>min</th>
<th>&lt;xs:element name=&quot;min&quot; type=&quot;vector&quot;/&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>The lower corner of the box</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>max</th>
<th>&lt;xs:element name=&quot;max&quot; type=&quot;vector&quot;/&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>The upper corner of the box</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iscutter</th>
<th>&lt;xs:element name=&quot;iscutter&quot; type=&quot;xs:boolean&quot;/&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whether this object is a cutter (set to true) or a normal object (set to false).</td>
<td></td>
</tr>
</tbody>
</table>

Group: Motions
<xs:group ref="motion_group" minOccurs="0" maxOccurs="unbounded"/>
Local motions for the box. See Section A.4.5.

A.4.4.3  <cylinder>

Again like Sphere and Box, Cylinder is an analytically defined circular-cylinder object used for testing, basic object composition, and as a cutter. A cylinder is created centered at the origin and aligned with the Z axis. Translations and
rotations may be used to position it however desired.

<table>
<thead>
<tr>
<th>name</th>
<th>&lt;xs:element name=&quot;name&quot; type=&quot;xs:string&quot; minOccurs=&quot;1&quot; maxOccurs=&quot;1&quot;/&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>A name for the cylinder.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>radius</th>
<th>&lt;xs:element name=&quot;radius&quot; type=&quot;xs:double&quot; minOccurs=&quot;1&quot; maxOccurs=&quot;1&quot;/&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>The cylinder radius, in the x-y plane.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>length</th>
<th>&lt;xs:element name=&quot;length&quot; type=&quot;xs:double&quot; minOccurs=&quot;1&quot; maxOccurs=&quot;1&quot;/&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>The cylinder length, in the z direction, centered at the origin.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iscutter</th>
<th>&lt;xs:element name=&quot;iscutter&quot; type=&quot;xs:boolean&quot;/&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whether this object is a cutter (set to true) or a normal object (set to false).</td>
<td></td>
</tr>
</tbody>
</table>

**Group: Motions**

<table>
<thead>
<tr>
<th>&lt;xs:group ref=&quot;motion_group&quot; minOccurs=&quot;0&quot; maxOccurs=&quot;unbounded&quot;/&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local motions for the cylinder. See Section A.4.5.</td>
</tr>
</tbody>
</table>

**A.4.4.4 <stlobject>**

An STL object is a triangulated surface typically output from a CAD package. As a discrete geometry the internal handling for an STL object is substantially more complex than for an analytical object, requiring significantly more time and more memory for handling. IBSEN has support for resolving some common types of CAD-file-output degeneracies, and is relatively insensitive to there degeneracies, but highly complex geometries may still require some manual cleanup prior to use in IBSEN. Be sure to check the flow solution frequently to ensure no hidden problems emerge during the run, and note that for best vi-
sualization of results the input STL file should be a “wetted surface”—that is, a closed surface with no points lying inside the solid. The solution procedure may be adversely affected if it is not (though in many cases it will run without any problems).

<table>
<thead>
<tr>
<th>name</th>
<th>&lt;xs:element name=&quot;name&quot; type=&quot;xs:string&quot;/&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>A name for the STL object.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>file</th>
<th>&lt;xs:element name=&quot;file&quot; type=&quot;xs:string&quot;/&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>The file to read the geometry from.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>zones</th>
<th>&lt;xs:element name=&quot;zones&quot; type=&quot;int_vector&quot; minOccurs=&quot;0&quot; maxOccurs=&quot;1&quot;/&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>A vector element listing out which zones to read from the STL file. The zones are numbered from 1. In many cases it is best to use the XML file to break up a complex STL object into multiple, simpler subobjects, rather than reading them all into a single object.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>momentpoint</th>
<th>&lt;xs:element name=&quot;momentpoint&quot; type=&quot;vector&quot;/&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>A point to calculate the moment about for the force and moment calculations.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>asciiflag</th>
<th>&lt;xs:element name=&quot;asciiflag&quot; type=&quot;xs:boolean&quot;/&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set to “true” if the input file is ASCII, or “false” if it is binary. IBSEN supports both types of STL input.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iscutter</th>
<th>&lt;xs:element name=&quot;iscutter&quot; type=&quot;xs:boolean&quot;/&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whether this object is a cutter (set to true) or a normal object (set to false).</td>
<td></td>
</tr>
</tbody>
</table>
A Plot3D object is another type of discrete geometry input typical of many CFD packages. IBSEN supports multi-zone input of ASCII structures Plot3D surface grid files. The zones in a file must be non-overlapping and taken together must form a watertight, closed non-self-intersecting surface.

**name**

```xml
<xs:element name="name" type="xs:string"/>
```

A name for the Plot3D object.

**file**

```xml
<xs:element name="file" type="xs:string"/>
```

The file to read the geometry from.

**momentpoint**

```xml
<xs:element name="momentpoint" type="vector"/>
```

A point to calculate the moment about for the force and moment calculations.

**asciiflag**

```xml
<xs:element name="asciiflag" type="xs:boolean"/>
```

True if the file is ASCII, or false if it is double-precision Fortran UNFORMATTED. Raw binary or single-precision UNFORMATTED inputs are not currently supported. The file must be a 3D whole multi-zone file, even if it only contains a single zone.
iscutter

<xs:element name="iscutter" type="xs:boolean"/>
Whether this object is a cutter (set to true) or a normal object (set to false).

**Group: Elastic deformations**

<xs:group ref="elastic_deformation_group" minOccurs="0" maxOccurs="unbounded"/>
Deformations of the Plot3D object. See Section A.4.6.

**Group: Motions**

<xs:group ref="motion_group" minOccurs="0" maxOccurs="unbounded"/>
Local motions for the Plot3D object. See Section A.4.5.

### A.4.5 Motions

Nearly all entities in IBSEN support motion inputs: these motions are all linear rigid-body transformations and can be combined in any order to yield highly complex rigid-body motions for objects, grids, massless particle release points, and acoustic data surfaces. Objects may be linked together or allowed to move separately, all through the motion interface. Anywhere the XSD file specifies a <xs:group ref="motion_group">, any number of these motions may be included in that location of the XML file:

**aperiodictranslation**

<xs:element name="aperiodictranslation" type="aperiodic_translation_type"/>

**periodictranslation**

<xs:element name="periodictranslation" type="periodic_translation_type"/>
The inputs to each type of motion are as follows:

### A.4.5.1 `<aperiodictranslation>`

An aperiodic translation is used to represent a translation of the form 
\[ \mathbf{x}(t) = \mathbf{x}_0 + \mathbf{v}t + \frac{1}{2} \mathbf{a} t^2 \]

where \( \mathbf{x}_0 \) is a static (non-time-dependent) translation, \( \mathbf{v} \) is a velocity, and \( \mathbf{a} \) an acceleration. All quantities are input as vectors.

**name**

<xs:element name="name" type="xs:string" minOccurs="1" maxOccurs="1" />

The name of the translation.

**position**

<xs:element name="position" type="vector" minOccurs="0" maxOccurs="1" />

The static translation component, i.e. the translation at \( t=0 \).

**velocity**

<xs:element name="velocity" type="vector" minOccurs="0" maxOccurs="1" />

The velocity at \( t=0 \).
A periodic translation represents a harmonic oscillation specified by a short Fourier series,
\[
\vec{x}(t) = \vec{A}_0 - \vec{A}_1 \cos(\psi + \psi_0) - \vec{B}_1 \sin(\psi + \psi_0) - \vec{A}_2 \cos(2[\psi + \psi_0]) - \vec{B}_2 \sin(2[\psi + \psi_0]).
\]
All coefficients are input as vectors.

The name of the translation.

The period of the translation.

The initial $\psi$ value.

The $\vec{A}_0$ coefficient.

The $\vec{A}_1$ coefficient.

The $\vec{A}_2$ coefficient.
The $\vec{B}_1$ coefficient.

The $\vec{B}_2$ coefficient.

A periodic rotation represented by a Fourier series of arbitrary length. Rotations are always specified about a single local-frame Cartesian axis (i.e. x, y or z). More complex rotations must be decomposed.

A name for the rotation.

The axis of rotation (local frame).

The period of the function (time units).

The initial $\psi$. Unit attribute may be “degrees” or “radians” (defaults to “radians”).

The initial angle. Unit attribute may be “degrees” or “radians” (defaults to “radians”).
A

<xs:element name="A" type="angle_vector" minOccurs="0" maxOccurs="1" />
A list of Fourier cosine coefficients. Unit attribute may be “degrees” or “radians” (defaults to “radians”).

B

<xs:element name="B" type="angle_vector" minOccurs="0" maxOccurs="1" />
A list of Fourier sine coefficients. Unit attribute may be “degrees” or “radians” (defaults to “radians”).

A4.5.4 <periodicrotationfromfile>

A periodic rotation input as a finite series from a file, wrapping around to the beginning when the end of the series is reached. Internally the series is converted to a periodic cubic B-spline, guaranteeing the continuity of the velocity and acceleration calculations. The input file is a simple two-column ASCII file with the first column the time, azimuthal location, etc. that the rotation is keyed to, and the second column is the actual angle, in whatever units is desired (the conversion factor to radians is a user input). The specified rotation values in the file do not need to be uniformly spaced in time/azimuth.

name

<xs:element name="name" type="xs:string" />
A name for the rotation.

axis

<xs:element name="axis" type="axis_character" />
The axis of rotation (local frame).

periodintimeunits

<xs:element name="periodintimeunits" type="xs:double" />
The period of the rotation (time units)
### periodinfileunits

<xs:element name="periodinfileunits" type="xs:double"/>

The period of the rotation (the units used in the file).

### psiattimezero

<xs:element name="psiattimezero" type="xs:double"/>

The initial $\psi$ value specified in file units.

### multipliertogetradians

<xs:element name="multipliertogetradians" type="xs:double"/>

If the angle in the file is not in radians, multiply by this value to convert to radians.

### filename

<xs:element name="filename" type="filename"/>

The name of (or path to) the file to load.

---

**A.4.5.5 <aperiodicrotation>**

An aperiodic rotation is a rotation specified by $\theta = \theta_0 + \omega t + \alpha t^2$. Note that at present this is a scalar equation: all rotations are specified around a local cardinal axis.

### name

<xs:element name="name" type="xs:string" minOccurs="1" maxOccurs="1"/>

A name for the rotation.

### axis

<xs:element name="axis" type="axis_character" minOccurs="1" maxOccurs="1"/>

The axis of rotation (local frame).

### angle

<xs:element name="angle" type="angleWithUnits" minOccurs="0" maxOccurs="1"/>

The initial rotation angle, $\theta_0$. Unit attribute may be “degrees” or “radians” (defaults to “radians”).
omega

<x:s:element name="omega" type="angularRateWithUnits" minOccurs="0" maxOccurs="1"/>

The rotational velocity, \( \omega \), rad/s. Unit attribute may be “deg/s”, “rad/s”, or “rpm”. Defaults to “rad/s”.

alpha

<x:s:element name="alpha" type="angularAccelerationWithUnits" minOccurs="0" maxOccurs="1"/>

The rotational acceleration, \( \alpha \). Units attribute may be “deg/s^2” or “rad/s^2”. Defaults to “rad/s^2”.

A.4.5.6 <copymotionfrom>

The last type of motion is a special rotation object that allows bodies to move in conjunction (or to share a base set of motions). The “CopyMotionFrom” motion takes as a single input the name of the object to copy the motions from. This includes all of that objects’ parent motions, so that the copied object simply inserts all of those motions into its motion list. Additional motions may be specified after a copymotionfrom object, though it is not recommended (for clarity’s sake).

A.4.6 Elastic Motions

IBSEN includes basic support for elastic motions. These motions are currently only supported by Plot3D objects and Curvilinear solvers (though the extension to other object types is relatively straightforward). In addition, only two types of elastic motion are currently implemented: bending and twisting, both using discrete inputs from a data file (the HART-II file format was used as the standard).
A.4.6.1  <discretebendingdeformation>

Discrete bending is input via a file that lists a series of stations and deflection distances, optionally as a function of time (or some time-like variable such as azimuth). These deflections are then used as the nodes in a cubic B-spline to provide deflection as a continuous function of beamwise position. Time interpolation is simple linear interpolation between the spline values at the bracketing time locations. Note that the deflections and station positions must be provided in grid units. The actual function to provide the deformation simply shears the basic input shape along the specified deformation and may result in ill-conditioned grids if large deformations are applied to a flow solver.

<table>
<thead>
<tr>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;xs:element name=&quot;name&quot; type=&quot;xs:string&quot; /&gt;</td>
</tr>
<tr>
<td>A name for the deflection.</td>
</tr>
</tbody>
</table>

If the deformation is periodic, both of these must be set:

<table>
<thead>
<tr>
<th>periodintimeunits</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;xs:element name=&quot;periodintimeunits&quot; type=&quot;xs:double&quot; minOccurs=&quot;1&quot; maxOccurs=&quot;1&quot; /&gt;</td>
</tr>
<tr>
<td>The period of the deflection inputs in non-dimensional time units (time is nondimensionalized by the reference length (typically 1) and speed of sound).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>periodinfileunits</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;xs:element name=&quot;periodinfileunits&quot; type=&quot;xs:double&quot; minOccurs=&quot;1&quot; maxOccurs=&quot;1&quot; /&gt;</td>
</tr>
<tr>
<td>The file’s time unit may be something other than non-dimensional time (often rotor azimuth, etc.) This is the period in those units (360 if the units are degrees, for example, or $2\pi$ if radians).</td>
</tr>
</tbody>
</table>

If the deformation is aperiodic, neither of those elements may be set. All deformations must also specify:
beamaxis
<xs:element name="beamaxis" type="axis_character" minOccurs="1" maxOccurs="1" />
"x", "y", or "z"; the axis of the beam.

deformationdirection
<xs:element name="deformationdirection" type="axis_character" minOccurs="1" maxOccurs="1" />
"x", "y", or "z"; The direction of deformation. Must not be equal to the beam axis.

filename
<xs:element name="filename" type="xs:string" minOccurs="1" maxOccurs="1" />
The file the deformations are loaded from. This file can take two formats depending on whether the deformation is periodic or static (aperiodic is not supported at this time). The first line in the file is expected to be a series of beamwise stations in grid units (e.g. not necessarily \(r/R\) locations, unless the grid is nondimensionalized by the rotor radius). If the deflection is static, then the next line is simply a list of the deflection at each of those stations. If the deflections are periodic, then each subsequent line begins with the current time (or time-like quantity, e.g. azimuth) and then lists the deflections at that time.

A.4.6.2  <discretetwistingdeformation>
Discrete twisting is similar to discrete bending, except that instead of defining a simple deflection, it specifies a twist angle about an arbitrary axis.

name
<xs:element name="name" type="xs:string" />
A name for the deflection.

If the deformation is periodic, both of these must be set:

periodintimeunits
<xs:element name="periodintimeunits" type="xs:double" minOccurs="1" maxOccurs="1" />
The period of the twist inputs in non-dimensional time units (time is nondimensionalized by the reference length (typically 1) and speed of sound).
**periodinfileunits**

```
<xs:element name="periodinfileunits" type="xs:double" minOccurs="1" maxOccurs="1"/>
```

The file’s time unit may be something other than non-dimensional time (often rotor azimuth, etc.) This is the period in those units (360 if the units are degrees, for example, or $2\pi$ if radians).

If the deformation is aperiodic, neither of those elements may be set. The twist occurs about an arbitrary axis defined by two points in the local object coordinate system (e.g. before any transformations are applied). This axis is defined by:

**twistaxisstart**

```
<xs:element name="twistaxisstart" type="vector" minOccurs="1" maxOccurs="1"/>
```

A point defining the start of the twist axis, and the origin of the parametric coordinate system defined along that axis.

**twistaxisend**

```
<xs:element name="twistaxisend" type="vector" minOccurs="1" maxOccurs="1"/>
```

A second point on the twist axis, used to define the axis along with the origin.

**raxis**

```
<xs:element name="raxis" type="axis_character" minOccurs="1" maxOccurs="1"/>
```

Although the twist itself can be about an arbitrary axis, the radial station the twist is defined at must correspond to one of the ordinal directions.

**radius**

```
<xs:element name="radius" type="xs:double" minOccurs="1" maxOccurs="1"/>
```

Twist is given as a function of $r/R$ position along the geometry: the coordinate system begins at the local coordinate system origin, and extends along the $raxis$ axis for this distance, which should correspond to $r/R = 1$. 
angle units
<xs:element name="angleunits" type="angleUnits" minOccurs="1" maxOccurs="1"/>
The input twist angle may be either “degrees” or “radians”, as specified by this value.

filename
<xs:element name="filename" type="xs:string" minOccurs="1" maxOccurs="1"/>
The file the deformations are loaded from. This file can take two formats depending on whether the deformation is periodic or static (aperiodic is not supported at this time). The first line in the file is expected to be a series of beamwise stations in $r/R$ coordinates. If the deflection is static, then the next line is simply a list of the twists at each of those stations. If the deflections are periodic, then each subsequent line begins with the current time (or time-like quantity, e.g. azimuth) and then lists the twists at that time.

A.4.6.3 <copydeformationfrom>
Analogous to the copymotionfrom element in the rigid-body motions, this element simply takes a string that is the name of an object above it in the XML input file. Any elastic motions specified for that object are copied to this one.

copydeformationfrom
<xs:element name="copydeformationfrom" type="xs:string"/>
The name of the object to copy the deformations from.

A.4.7 Solvers and Related Objects

In IBSEN, a “Solver” is a region of space with a grid, a differentiation scheme, and potentially surface and/or farfield boundary conditions. Each grid in an overset grid system is represented by a single Solver object, although many times the system is created automatically using an off-body grid generator (defined below). While IBSEN was designed to support multiple disparate types
of Solver, as of this writing, only two main types of solver are implemented: a uniform-grid Euler solver used for farfield regions, and a general curvilinear Euler solver for use in body-fitted grid applications (although both solvers support immersed boundaries). In the <case> input, the last specified input is any number of items from the “master_solver_group.” This group consists of the following elements:

**curvilineareuler**
<xs:element name="curvilineareuler" type="curvilineareuler_type" />
The second of the two current main solver classes, see Section A.4.7.1

**uniformeuler**
<xs:element name="uniformeuler" type="uniformeuler_type" />
One of two current main solver classes, see Section A.4.7.2

**generaloffbodygridgenerator**
<xs:element name="generaloffbodygridgenerator" type="generaloffbodygridgenerator_type" />
A general-purpose off-body grid generator (also capable of including the on-body grid if desired): the most robust and flexible of the grid generators. See Section A.4.7.3

**masslessparticlereleasepoint**
<xs:element name="masslessparticlereleasepoint" type="masslessparticlereleasepoint_type" />
A point to release massless particles from to track the vortex filaments in the solution. See Section A.4.7.4

**Group: Acoustic Data Surfaces**
<xs:group ref="acousticdatasurfacergroup" />
Any of a number of different types of acoustic data surface. See Section A.4.8
A.4.7.1 <curvilineareuler>

The <curvilineareuler> element defines the most general type of mesh supported by IBSEN: this is a general curvilinear structured grid with no particular relationship between the \((i,j,k)\) and \((x,y,z)\) coordinate spaces. This type of mesh is input from a Plot3D-formatted grid file, and supports the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>name</strong></td>
<td>A name for the solver.</td>
</tr>
<tr>
<td><strong>Group: Differentiators</strong></td>
<td>A single differentiation scheme must be specified: in the current version of IBSEN this must be the &lt;cd2/&gt; element, though others exist. See Section A.4.11.</td>
</tr>
<tr>
<td><strong>gridfile</strong></td>
<td>The path to the file to read the grid from. This file must be a double-precision Plot3D multi-zone Fortran-UNFORMATTED file. The grid must be defined with a right-handed coordinate system and must not have any negative Jacobians.</td>
</tr>
<tr>
<td><strong>zone</strong></td>
<td>The index of the zone to read from the specified grid file. The first zone is zone 1 (e.g. Fortran-indexing is used).</td>
</tr>
<tr>
<td><strong>fringethickness</strong></td>
<td>For an overset mesh, the number of layers at the outer edge of the mesh that are marked as receptors and seek donation from the rest of the solvers. This may be overridden at individual points or boundaries by setting a boundary condition there, in which case no donation is sought.</td>
</tr>
</tbody>
</table>
priority
<xs:element name="priority" type="xs:nonNegativeInteger" minOccurs="1" maxOccurs="1"/>
In oversetting calculations, grids with better priority take precedence over those with worse priority. The “highest” priority is 1: larger numbers indicate lower priority. There is no upper limit, and the specified priorities do not need to follow a linear sequence: any value may be specified. Typically near-body body-fitted meshes are all marked as priority 1, and other meshes are given higher numerical values (e.g. lower priority).

group
<xs:element name="group" type="xs:nonNegativeInteger" minOccurs="0" maxOccurs="1"/>
Optionally, solvers may be grouped together for optimization purposes: it is possible to force the code skip re-checking the donations between grids in the same group. For example, if a body-fitted rotor mesh system consists of three grids, one for the root, one for the tip, and one for the span, these grids are often placed in a single group, and that groups donations are marked as static (see below). This causes IBSEN to use the same interpolants in the overlapping regions between those three grids at every timestep, never re-evaluating them.

donationswithingrouparestatic
<xs:element name="donationswithingrouparestatic" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
Working in conjunction with the “group” specified above, setting this to true enables an optimization where the donations occurring within a single group are never re-evaluated, even if the group elements are moving.

isinterfacegrid
<xs:element name="isinterfacegrid" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
This is a critical parameter controlling the oversetting algorithms: any mesh that either moves through other meshes, or has other meshes move through it during the course of the run should have this set to true. Failure to do so will result in incorrect oversetting. Setting it to true on meshes that do not require it is safe, but will slow the run down.
**disablebodychecks**

```xml
<xs:element name="disablebodychecks" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
```

If it is known a priori that this mesh will never intersect an immersed boundary, disabling the body checks by setting this to true will speed up the run.

**disablebodychecksafterstartup**

```xml
<xs:element name="disablebodychecksafterstartup" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
```

If it is known a priori that the immersed boundary elements that this mesh intersects are static (e.g. the boundary has the same motions as the grid, and no other boundaries cross into the grid later in the run) setting this to true will speed up the run.

**epsilon2**

```xml
<xs:element name="epsilon2" type="xs:double" minOccurs="0"/>
```

IBSEN uses an ARC3D-style dissipation term: epsilon2 controls the second-order dissipation, and defaults to 2.0 (adequate for most rotor cases).

**epsilon4**

```xml
<xs:element name="epsilon4" type="xs:double" minOccurs="0"/>
```

IBSEN uses an ARC3D-style dissipation term: epsilon4 controls the fourth-order dissipation, and defaults to 0.04 (adequate for most rotor cases).

**dontsolvejusttest**

```xml
<xs:element name="dontsolvejusttest" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
```

A test purely for debugging, this parameter sets up a function in the solution array that allows testing to ensure the oversetting is working properly. Not useful for normal runs.

**Group: Surface Treatments**

```xml
<xs:group ref="surfacetreatment_group" minOccurs="1" maxOccurs="1"/>
```

If an immersed boundary is present, a surface treatment must be set. See section A.4.10
A.4.7.2  <uniformeuler>

The <uniformeuler> element defines a uniformly-spaced Cartesian-grid Euler flow solution region. A typical case consists of a number of these elements, each covering a specified region of space. Those that intersect the body generally have body treatment methods specified, and those in the farfield have farfield domain boundary conditions. At a minimum a minimum and maximum corner and a grid spacing is required.

The following inputs are supported:

**name**

<x:s:element name="name" type="xs:string"/>

A name for the solver.

---

**Group: Differentiators**

<x:group ref="differentiator_group" minOccurs="1" maxOccurs="1"/>

A single differentiation scheme must be specified: in the current version of IBSEN this must be the <cd2/> element, though others exist. See Section A.4.11.
min
<xs:element name="min" type="vector" minOccurs="1" maxOccurs="1"/>
The minimum corner of the bounding box, in local coordinates.

max
<xs:element name="max" type="vector" minOccurs="1" maxOccurs="1"/>
The maximum corner of the bounding box, in local coordinates.

spacing
<xs:element name="spacing" type="vector" minOccurs="1" maxOccurs="1"/>
The grid spacing in each direction.

fringethickness
<xs:element name="fringethickness" type="xs:nonNegativeInteger" minOccurs="0" maxOccurs="1"/>
For an overset mesh, the number of layers at the outer edge of the mesh that are marked as receptors and seek donation from the rest of the solvers. This may be overridden at individual points or boundaries by setting a boundary condition there, in which case no donation is sought.

priority
<xs:element name="priority" type="xs:nonNegativeInteger" minOccurs="1" maxOccurs="1"/>
In oversetting calculations, grids with better priority take precedence over those with worse priority. The “highest” priority is 1: larger numbers indicate lower priority. There is no upper limit, and the specified priorities need to follow a linear sequence: any value may be specified. Typically near-body body-fitted meshes are all marked as priority 1, and other meshes are given higher numerical values (e.g. lower priority).

group
<xs:element name="group" type="xs:nonNegativeInteger" minOccurs="0" maxOccurs="1"/>
Optionally, solvers may be grouped together for optimization purposes: it is possible to force the code to skip re-checking the donations between grids in the same group. For example, if a body-fitted rotor mesh system consists of three grids, one for the root, one for the tip, and one for the span, these grids are often placed in a single group, and that groups donations are marked as static (see below). This causes IBSEN to use the same interpolants in the overlapping regions between those three grids at every timestep, never re-evaluating them.
**donationswithingrouparestatic**

```xml
<xs:element name="donationswithingrouparestatic" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
```

Working in conjunction with the “group” specified above, setting this to true enables an optimization where the donations occurring within a single group are never re-evaluated, even if the group elements are moving.

**isinterfacegrid**

```xml
<xs:element name="isinterfacegrid" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
```

This is a critical parameter controlling the oversetting algorithms: any mesh that either moves through other meshes, or has other meshes move through it during the course of the run should have this set to true. Failure to do so will result in incorrect oversetting. Setting it to true on meshes that do not require it is safe, but will slow the run down.

**noslip**

```xml
<xs:element name="no slip" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
```

Deprecated. For testing various immersed boundary methods.

**filter**

```xml
<xs:element name="filter" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
```

Deprecated. A sixth-order filter for suppressing oscillations. Should not be necessary with the cd2 differentiator and ARC3D dissipation.

**disablebodychecks**

```xml
<xs:element name="disablebodychecks" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
```

If it is known a priori that this mesh will never intersect an immersed boundary, disabling the body checks by setting this to true will speed up the run.

**disablebodychecksafterstartup**

```xml
<xs:element name="disablebodychecksafterstartup" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
```

If it is known a priori that the immersed boundary elements that this mesh intersects are static (e.g. the boundary has the same motions as the grid, and no other boundaries cross into the grid later in the run) setting this to true will speed up the run.
IBSEN uses an ARC3D-style dissipation term: epsilon2 controls the second-order dissipation, and defaults to 2.0 (adequate for most rotor cases).

IBSEN uses an ARC3D-style dissipation term: epsilon4 controls the fourth-order dissipation, and defaults to 0.04 (adequate for most rotor cases).

A test purely for debugging, this parameter sets up a function in the solution array that allows testing to ensure the oversetting is working properly. Not useful for normal runs.

If the cutters are not moving with respect to this mesh, setting this to true will speed up the run. See section A.4.7.2.1.

A list of the cutters to use. See section A.4.7.2.1.

If an immersed boundary is present, a surface treatment must be set. See section A.4.10

If any of this solver’s boundaries are not set by grid oversetting, boundary conditions must be set at those location. See section A.4.9.
Group: Elastic Deformations
<xs:group ref="elastic_deformation_group" minOccurs="1" maxOccurs="1"/>
If the grid deforms with time, or has a static elastic deformation applied to it before the run begins, this section lists those deformations. See section A.4.6.

Group: Rigid-Body Motions
<xs:group ref="motion_group" minOccurs="1" maxOccurs="1"/>
The rigid-body solver motions. See section A.4.5.

A.4.7.2.1 Cutters  Uniform Eulerian meshes are designed to form the background mesh system for body-fitted curvilinear meshes. During the oversetting procedure points in the uniform meshes may lie inside the region of space occupied by the body: the body-fitted grids are unable to interpolate to these positions, and because the body is covered by a body-fitted mesh no immersed boundary is available. Since these are not valid flow points they need to be “blanked” out of the solution procedure. To do this, objects are created in the main object input section of the input that are specifically designated as “cutters.” These objects do not participate in the flow solution as immersed boundaries, and no data is ever interpolated onto them: the serve only to designate a region of space as unavailable for flow solution. Any object type supported by IBSEN may be a cutter: for a rotor blade the simplest is to use a box element just slightly larger than the rotor blade. Another possibility is to use the second layer of gridpoints from the curvilinear mesh defining the blade region. It is critical that this cutter object completely enclose the region of space where the body-fitted meshes are unable to interpolate, but does not extend beyond the outer boundaries of those meshes.

To specify a list of cutters in a Uniform Euler solver, the format is:
<cutterstouse>
  <data>Cutter 1</data>
  <data>Another cutter</data>
  <data>Etc.</data>
</cutterstouse>

where “Cutter 1”, “Another cutter”, and “Etc.” are the names specified for the cutters in the physical objects section. Note that the object must be marked as a cutter or it will be ignored, but any type of object may be marked as a cutter.

In most cases, objects such as rotor blades will be specified twice in the physical object section: once as a real object for surface pressure calculation and once as a cutter to eliminate interior points from the background Cartesian meshes.

A.4.7.3 <generaloffbodygridgenerator>

This grid generator creates a system of successively coarser uniform Cartesian grids out to a user-defined limit, greatly simplifying the task of generating complex three-dimensional overset grid systems. A typical helicopter case setup has a set of curvilinear Euler body-fitted meshes for the rotor blades, and then a single off-body grid generator used to create the mesh system enclosing those blades and extending to the farfield. It is possible to have multiple off-body grid generators, but this is rarely used.

In the internal implementation, the general off-body grid generator creates a set of UniformEuler objects, so in addition to the generator-specific inputs, all of the standard inputs to UniformEuler are accepted (see below).

<table>
<thead>
<tr>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;xs:element name=&quot;name&quot; type=&quot;xs:string&quot;/&gt;</td>
</tr>
<tr>
<td>A name for the grid generator: this is used to construct the names of the individual UniformEuler objects.</td>
</tr>
</tbody>
</table>
innergridmincorner
<xs:element name="innergridmincorner" type="vector" minOccurs="1" maxOccurs="1"/>
The lower corner of the level 1 brick (innermost grid). Note that this must be specified even if the innermost grid is not being automatically generated.

innergridmaxcorner
<xs:element name="innergridmaxcorner" type="vector" minOccurs="1" maxOccurs="1"/>
The upper corner of the level 1 brick (innermost grid). Note that this must be specified even if the innermost grid is not being automatically generated.

innerspacing
<xs:element name="innerspacing" type="vector" minOccurs="1" maxOccurs="1"/>
The spacing of the level 1 brick (innermost grid). This spacing is doubled at each successive grid (unless overwritten by the stopdoublingafterlevel parameter).

createinnergrid
<xs:element name="createinnergrid" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
Whether or not to actually create the innermost grid. Note that even if this grid is not created, it is still treated as the level 1 brick for the purposes of determining the position and spacing of the other meshes.
This is a relatively complex input for this grid generator: this sets the number of levels to expand in each direction, specified as three sets of (min,max) pairs. The input looks like:

```xml
<nblevels>
  <i><first>1</first><second>2</second></i>
  <j><first>3</first><second>4</second></j>
  <k><first>5</first><second>6</second></k>
</nblevels>
```

where “first” is the negative direction and “second” is the positive direction. This allows complete flexibility in terms of the grids’ growth (at the expense of input complexity, of course).

Similar to nblevels, this input sets the number of points to expand in each direction, per level. It’s input looks just like the nblevels input.

In each direction, specify the maximum number of spacing doublings. This allows you to stop the spacing increases but still add additional layers to the grid (for example, if you wanted to stop doubling in just one direction, but continue doubling in the others).

The number of overset fringe points to use. Currently must be at least 2.

The priority of the generated solvers.
interfacelevels
<xs:element name="interfacelevels" type="int_vector" minOccurs="0" maxOccurs="1" />
A list of the levels that are expected to overset with grids moving relative to them. A simple space-separated list of integers. The innermost grid is level 0.

disablebodycheckslevels
<xs:element name="disablebodycheckslevels" type="int_vector" minOccurs="0" maxOccurs="1" />
It is possible to selectively disable the immersed boundary intersection tests on a level-by-level basis using this parameter.

disablebodychecksafterstartuplevels
<xs:element name="disablebodychecksafterstartuplevels" type="int_vector" minOccurs="0" maxOccurs="1" />
This allows the specified solvers to check for immersed boundaries at the first timestep, and then treats all of those bodies as static as the solution proceeds. This is an optional optimization, but can greatly speed up runs where the immersed boundaries are not moving relative to the grids.

outputxml
<xs:element name="outputxml" type="xs:boolean" minOccurs="0" maxOccurs="1" />
Internally an XML document is created to set up these grids: this document can be written to standard output for examination, and possible direct inclusion in a later run.

The following inputs are passed directly to all of the uniformeuler objects created by this grid generator: for their definitions see Section A.4.7.2.

Group: Differentiators
<xs:group ref="differentiator_group" minOccurs="1" maxOccurs="1" />
disablebodychecks
<xs:element name="disablebodychecks" type="xs:boolean" minOccurs="0" maxOccurs="1"/>

disablebodychecksafterstartup
<xs:element name="disablebodychecksafterstartup" type="xs:boolean" minOccurs="0" maxOccurs="1"/>

isinterfacegrid
<xs:element name="isinterfacegrid" type="xs:boolean" minOccurs="0" maxOccurs="1"/>

group
<xs:element name="group" type="xs:nonNegativeInteger" minOccurs="0" maxOccurs="1"/>

donationswithingrouparestatic
<xs:element name="donationswithingrouparestatic" type="xs:boolean" minOccurs="0" maxOccurs="1"/>

dontsolvejusttest
<xs:element name="dontsolvejusttest" type="xs:boolean" minOccurs="0" maxOccurs="1"/>

cuttersarestatic
<xs:element name="cuttersarestatic" type="xs:boolean" minOccurs="0" maxOccurs="1"/>

cutterstouse
<xs:element name="cutterstouse" type="string_vector" minOccurs="0" maxOccurs="1"/>

Group: Immersed boundary conditions
<xs:group ref="surfacetreatment_group" minOccurs="0" maxOccurs="1"/>

Next, in most cases all six sides of the resulting domain require boundary conditions: this section takes one boundary condition (of any type IBSEN recognizes) for each boundary. In general it has been found that the Buffer Zone
boundary condition provides the most stable results when used in conjunction with relatively large solution domains.

Finally, it is possible to specify motions for the background meshes: all motions that IBSEN recognizes are valid here, although the most common case type simply specifies a stationary background mesh system.
A.4.7.4  \texttt{<masslessparticlerleasepoint>}

A massless-particle release point is a position in space, possibly moving, from which massless line-vortex tracking particles are released. These particles work together with the acoustic data surfaces to improve the capturing of BVI noise. They can also be used for simple vortical flow visualization. They \textbf{do not} affect the flow solution.

There are two ways to specify the particle release position: the first, and simplest, is to specify a single explicit position to release the particles at. This is most useful in cases with a static vorticity generating surface, where the core position of the vortex is known a priori. For the more general case, where the position of the core is expected to vary with time, a “release surface” can be specified. This surface is a finite-sized circular region of space whose vorticity is examined at each release time. The actual seeding position for the particle is chosen to be the location of the peak vorticity on that circular plane.

\begin{verbatim}
name
  <xs:element name="name" type="xs:string"/>
  The name of the release point.
\end{verbatim}

You must either set an explicit position, or define a small search region. If you define a search region (by definition a circular plane with a given center, radius, and normal) the vorticity on that plane is examined and the position with either the highest vorticity magnitude, or the highest vorticity dot plane vector, is chosen as the release location.

To specify a single discrete release position:

\begin{verbatim}
position
  <xs:element name="position" type="vector" minOccurs="1" maxOccurs="1"/>
  The release position to use.
\end{verbatim}
To specify a circular region of space to search for the best search position:

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| **searchplanecenter**           | <xs:element name="searchplanecenter" type="vector" minOccurs="1" maxOccurs="1" />  
The center of the circular region to search. |
| **searchplanenormal**           | <xs:element name="searchplanenormal" type="vector" minOccurs="1" maxOccurs="1" />  
The normal vector of the plane on which to search. |
| **searchradius**                | <xs:element name="searchradius" type="xs:double" minOccurs="1" maxOccurs="1" />  
The radius from the center to end the search at. Because a finite number of points are examined on the release plane, the smaller this radius is the more accurately the particle will be seeded (limited by the accuracy of the solution field’s spatial resolution, and assuming that the location of peak vorticity lies within the radius specified). |
| **dotvorticitywithplane**       | <xs:element name="dotvorticitywithplane" type="xs:boolean" minOccurs="0" maxOccurs="1" />  
When choosing the seed position on the plane, the code defaults to using the magnitude of the vorticity vector. In particularly chaotic flows it may be better to use the vorticity vector dotted with the plane normal instead: note that the direction of the plane normal is important here, as the particles are seeded at the position this function maximizes—no absolute value is taken |
| **outputfile**                  | <xs:element name="outputfile" type="filename" minOccurs="0" maxOccurs="1" />  
The name of the file to write this vortex to. Note that this must be unique, and cannot be shared with other release points. IBSEN does not check this, however. |
**restartfile**

```xml
<xs:element name="restartfile" type="filename" minOccurs="0" maxOccurs="1"/>
```

If a restart file is specified here, that file is read in (even if the main case is not a restart). It is searched for the current timestep, which may be zero, and the release point is initialized with the particles from that step. The only exception to this is that for convenience, if the current timestep is zero, and step zero is not found in the file, the case silently starts with no particles.

**forcestatic**

```xml
<xs:element name="forcestatic" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
```

If set to true, the particles will not get updated during the run, nor will new particles be seeded. This is useful for types of cases where a line of particles is input and not expected to evolve with time.

**integrationradius**

```xml
<xs:element name="integrationradius" type="xs:double" minOccurs="1" maxOccurs="1"/>
```

When calculating the circulation, this is the radius of the circle that is integrated over. Note that this is completely independent of the search radius when the particle is seeded on a release plane: the radius specified here is centered on the particle itself, and the integration path is a circle on the plane formed by that point and a vector pointing in the direction of the local vorticity.

**viscouscoreradius**

```xml
<xs:element name="viscouscoreradius" type="xs:double" minOccurs="0" maxOccurs="1"/>
```

The numerical model for the vortex during the Biot-Savart integration includes a viscous core region to eliminate the infinite induced velocity at the core of a pure potential vortex. This parameter defines the size of the viscous core region. It defaults to 0.1.
circulationtype
<xs:element name="circulationtype" type="circulationType" minOccurs="0" maxOccurs="1"/>

When calculating the induced velocity there are three different circulation \( \Gamma \) values that can be used:

**peak** The peak circulation ever encountered by this particle as it travels through the field. Used with the vortex strength is expected to be maintained throughout the roll-up process.

**initial** The circulation at the particles initial release time and position. Rarely used, as the particle is usually released too close to the vorticity-generating surface for the vortex to have completely rolled up yet.

**bound** The maximum bound circulation on a specified vorticity-generating surface at the time the particle is released. The parameters of this surface are described below.

If using the bound circulation, these three must be set: the file is a Fortran-unformatted multi-zone plot3D surface mesh. Zone numbering uses the Fortran convention, starting at 1 (rather than zero). The same is true for integration direction.

surfacedefinitionfile
<xs:element name="surfacedefinitionfile" type="filename" minOccurs="0" maxOccurs="1"/>

The filename of a Plot3D file containing a zone that describes the vorticity-generating surface.

surfacedefinitionzone
<xs:element name="surfacedefinitionzone" type="xs:nonNegativeInteger" minOccurs="0" maxOccurs="1"/>

The number of the zone in that file. The first zone is zone one (not zero).

surfaceintegrationdirection
<xs:element name="surfaceintegrationdirection" type="direction_type" minOccurs="0" maxOccurs="1"/>

The index over which to traverse to calculate the bound circulation, \( i \) or \( j \).
**usecirculationdifference**

```xml
<xs:element name="usecirculationdifference" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
```

If set to true, uses the difference between the current circulation and the desired circulation (if the two are the same sign and the desired is larger than the current, otherwise the particle is ignored).

**distancecutoff**

```xml
<xs:element name="distancecutoff" type="xs:double" minOccurs="0" maxOccurs="1"/>
```

When calculating the induced velocity you can optionally exclude filaments that are too far from the point in question. To do that, set a distance cutoff here, in grid units.

**releasetimecontroller**

```xml
<xs:element name="releasetimecontroller" type="intermittentcontroller_type" minOccurs="0" maxOccurs="1"/>
```

The controller defining when a new particle is released.

**writetimecontroller**

```xml
<xs:element name="writetimecontroller" type="intermittentcontroller_type" minOccurs="0" maxOccurs="1"/>
```

The controller defining when the particles are written out. Independent of (and not related to) the visualization write controller in CFDRunner, which causes the particles to be written in a VTK file format not suitable for restarting, only for visualization.

**Group: Motions**

```xml
<xs:group ref="motion_group" minOccurs="0" maxOccurs="unbounded"/>
```

The release point/surface motions. See section A.4.5.

### A.4.8 Acoustic Data Surfaces

Acoustic data surfaces are used to write flow data out in PSU-WOPWOP format. At this time, two types of surfaces are supported: permeable cylinders and impermeable surfaces.
Permeable cylinders are designed to surround sound-generating bodies and to capture any noise generated by the surface of the body or by the flow contained within the permeable surface. They may move with the body or may remain stationary (keeping in mind that PSU-WOPWOP cannot handle surfaces whose speed is greater than or equal to the local speed of sound).

**name**

```xml
<xs:element name="name" type="xs:string"/>
```

A name for the ADS.

**radius**

```xml
<xs:element name="radius" type="xs:double"/>
```

The radius of the cylinder.

**startpoint**

```xml
<xs:element name="startpoint" type="vector"/>
```

The startpoint of the cylinder in local coordinates.

**stoppoint**

```xml
<xs:element name="stoppoint" type="vector"/>
```

The endpoint of the cylinder in local coordinates.

**pointsacrossradius**

```xml
<xs:element name="pointsacrossradius" type="xs:nonNegativeInteger"/>
```

The number of points across the radius for the endcap output.

**pointsalonglength**

```xml
<xs:element name="pointsalonglength" type="xs:nonNegativeInteger"/>
```

The number of points along the length of the cylinder.

**pointsaroundcircumference**

```xml
<xs:element name="pointsaroundcircumference" type="xs:nonNegativeInteger"/>
```

The number of points around the circumference, keeping in mind that the first and last points will repeat to ensure a closed surface.
**periodic**

```xml
<x:s:element name="periodic" type="xs:boolean"/>
```

Whether to treat the surface as periodic or not. If true, the first and last output points must be at the same point in the period.

**pressureunitstring**

```xml
<x:s:element name="pressureunitstring" type="psu-wopwop_acoustic_pressure_units"/>
```

A string representing the units of pressure in this case. Possibilities are enumerated below.

**outputcontroller**

```xml
<x:s:element name="outputcontroller" type="interrmtntcontroller_type"/>
```

A controller governing when to write the acoustics data. See Section A.4.3.

**Group: Motions**

```xml
<x:s:group ref="motion_group" minOccurs="0" maxOccurs="unbounded"/>
```

The ADS motions. See section A.4.5.

### A.4.8.2 `<surfacepressureads>`

Surface pressure acoustic data surfaces allow the output of either body surface pressure, or compact loading.

**name**

```xml
<x:s:element name="name" type="xs:string"/>
```

A name for the ADS.

**inputsurfacefilename**

```xml
<x:s:element name="inputsurfacefilename" type="xs:string"/>
```

The input Plot3D-formatted surface geometry file. Multi-zone, ASCII, no iBlanks, kMax=1.

**periodic**

```xml
<x:s:element name="periodic" type="xs:boolean"/>
```

Whether to treat the surface as periodic or not. If true, the first and last output points must be at the same point in the period.
**pressureunitstring**
<xs:element name="pressureunitstring" type="psu-wopwop_acoustic_pressure_units"/>
A string representing the units of pressure in this case. Possibilities are enumerated below.

**usemasslessparticles**
<xs:element name="usemasslessparticles" type="xs:boolean"/>
True to use the massless particles to augment the pressure.

**useflowfieldpressure**
<xs:element name="useflowfieldpressure" type="xs:boolean"/>
True to include the flowfield pressure. Note that if this and the massless particles flag are true, the quantities are summed.

**createcompactpatch**
<xs:element name="createcompactpatch" type="xs:boolean"/>
True to create a compact patch, or false to use distributed surface pressure. A compact patch can only be created for the first zone in the plot3D input file, and is created at the quarter chord. It is assumed that the plot3D file defines a surface that starts at the trailing edge and wraps around the leading edge before returning to the trailing edge again, repeating the last point.

**outputcontroller**
<xs:element name="outputcontroller" type="intermittentcontroller_type"/>
A controller governing when to write the acoustics data. See Section A.4.3.

**Group: Motions**
<xs:group ref="motion_group" minOccurs="0" maxOccurs="unbounded"/>
The ADS motions. See section A.4.5.

### A.4.9 Farfield Boundary Conditions

#### A.4.9.1 <bufferzone>
A buffer zone boundary condition is based on Wasistho’s work in Ref. 139. It slowly drives the solution at the boundaries to the freestream, allowing no
quantities to “float.” This boundary condition has one input:

```
bufferthickness
<xs:element name="bufferthickness" type="xs:integer"/>
```

The number of layers to buffer the solution.

A.4.9.2  `<subsonicinflow>`

A subsonic outflow boundary condition is based on the method of characteristics: at an outflow boundary, there are four outgoing characteristics, so one quantity (the pressure) is set from the freestream. The others are extrapolated from the interior. Note that this is not a non-reflecting boundary condition and tends to reflect strong pressure waves back into the domain. It has no inputs, so is specified in the XML file as “`<subsonicinflow/>`”.

A.4.9.3  `<subsonicoutflow>`

A subsonic inflow boundary condition is based on the method of characteristics: at an inflow boundary, there is one outgoing characteristic, so one quantity (the pressure) is extrapolated from the interior. The others are set to the freestream. Note that this is not a non-reflecting boundary condition and tends to reflect strong pressure waves back into the domain. It has no inputs, so is specified in the XML file as “`<subsonicoutflow/>`”.

A.4.9.4  `<slipwall>`

A slip wall boundary condition sets the velocity component perpendicular to the boundary to zero, extrapolating the remaining quantities. It has no inputs, so is specified in the XML file as “`<slipwall/>`”.
A.4.10 Surface Treatments

Several surface treatment methods have been implemented in IBSEN for testing purposes. It is anticipated that in the future further techniques will be added as research progresses, but at present it is strongly recommended that for general code runs, the inverse distance weighted average technique be used.

A.4.10.1 <inversedistanceweightedaveragereconstruction>

The most stable reconstruction method for use with coarse grids and slip-wall surface boundary conditions (as specified in the Solver inputs above). This type of surface boundary condition has two optional inputs:

<table>
<thead>
<tr>
<th>minpoints</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;xs:element name=&quot;minpoints&quot; type=&quot;xs:nonNegativeInteger&quot; minOccurs=&quot;0&quot;/&gt;</code></td>
</tr>
<tr>
<td>The minimum number of points in the average. Defaults to 5.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rescalemomentum</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;xs:element name=&quot;rescalemomentum&quot; type=&quot;&quot; minOccurs=&quot;0&quot;/&gt;</code></td>
</tr>
<tr>
<td>Whether to maintain a constant momentum magnitude when the surface boundary condition is applied. Defaults to false.</td>
</tr>
</tbody>
</table>

A.4.10.2 <surfacenormalextrapolationreconstruction>

A more sophisticated but less stable boundary condition, best suited to relatively fine grids or regions with low flowfield gradients. This technique has three optional inputs:

<table>
<thead>
<tr>
<th>order</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;xs:element name=&quot;order&quot; type=&quot;xs:nonNegativeInteger&quot; minOccurs=&quot;0&quot;/&gt;</code></td>
</tr>
<tr>
<td>The polynomial order for the extrapolation. Defaults to 1.</td>
</tr>
</tbody>
</table>
**usecurvature**

```xml
<xs:element name="usecurvature" type="""" minOccurs="0"/>
```

Whether to use the curvature of the surface in the pressure extrapolation. Defaults to false.

**rescalemomentum**

```xml
<xs:element name="rescalemomentum" type="""" minOccurs="0"/>
```

Whether to maintain a constant momentum magnitude when the surface boundary condition is applied. Defaults to false.

### A.4.10.3 <discreteforcingpenalization>

Forces the momentum at all interior points to zero. No inputs.

### A.4.10.4 <volumefractionpenalization>

Forces the momentum at all interior points to a fractional value depending on the amount of the cell immersed in the body. No inputs.

### A.4.10.5 <mirroringpenalization>

Forces the momentum at some number of interior layers to the opposite of the momentum at a point opposite it across the boundary. One optional input:

**mirrorlayerthickness**

```xml
<xs:element name="mirrorlayerthickness" type="xs:nonNegativeInteger" minOccurs="0"/>
```

The number of points to mirror. Defaults to 1.

### A.4.11 Differentiators

IBSEN implements a number of differentiators. At this time only the second-order central scheme is fully supported by all solvers, though support for the
others has existed in the past and could easily be added again. The body surface
treatment needs additional special cases to properly handle all of the schemes.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>weno5</td>
<td>A fifth-order WENO scheme, Lax-Friedrichs flux splitting.</td>
</tr>
<tr>
<td>uw1</td>
<td>First-order upwind, Lax-Friedrichs flux splitting.</td>
</tr>
<tr>
<td>uw3</td>
<td>Third-order upwind-biased, Lax-Friedrichs flux splitting.</td>
</tr>
<tr>
<td>uw5</td>
<td>Fifth-order upwind-biased, Lax-Friedrichs flux splitting.</td>
</tr>
<tr>
<td>cd2</td>
<td>Second-order central, implicit artificial dissipation.</td>
</tr>
</tbody>
</table>

### A.5 Running IBSEN

The executable for IBSEN is called “ibsen” and is meant to be run from the
command line, with a single argument, like this:

```
prompt$> ibsen InputDeck.xml
```

or, if you installed the executable someplace where it is not in your $PATH,

```
prompt$> /full/path/to/ibsen InputDeck.xml
```

IBSEN will then run on the local processor:
#!/bin/sh

# Cluster queuing system commands
PBS -N SemiInfiniteWing
PBS -l nodes=9:ppn=4

cd $PBS_O_WORKDIR
NPROCS=`wc -l < $PBS_NODEFILE`

# Run IBSEN and append the output to the file OUTPUT.txt
/usr/local/mvapich-1.0.1/bin/mpirun \
    -np $NPROCS \ 
    -machinefile $PBS_NODEFILE \ 
    ./ibsen SemiInfiniteWing.xml &> OUTPUT.txt

Figure A.1: Example input script for an MPI run on a cluster. Note that clusters vary and it is likely your script will look somewhat different.

If using MPI, you most likely need to submit your IBSEN job using a script run by your cluster’s queueing system, and to call the program via either “mpiexec” or “mpirun” depending on your setup. Talk to your cluster administrator if you do not know the proper method for your cluster. As an example, on one of the clusters at Penn State, a job script might look like that shown in Fig. A.1. See your cluster’s documentation for the location and syntax of your “mpirun” or “mpiexec” calls, as well as the queueing system commands.

As the code runs it prints out a lot of information as it sets the case up: storing this information in a file alongside your results can help you recreate the case later, as well as understand any problems you may run into during
% tail -f OUTPUT.txt
Writing reference (initial condition) restart file at t=0.
[dt=1.00e-06 (Δ = +174.61%)] Timestep 1 complete.
[dt=2.31e-06 (Δ = +18.54%)] Timestep 2 complete.
[dt=2.63e-06 (Δ = +3.76%)] Timestep 3 complete.
[dt=2.70e-06 (Δ = +0.59%)] Timestep 4 complete.
[dt=2.72e-06 (Δ < 0.5%)] Timestep 5 complete. Restart file written.
[dt=2.72e-06 (Δ < 0.5%)] Timestep 6 complete.

Figure A.2: Using the “tail” command to follow along with the output. The “-f” will continuously refresh the output as the file is written to.

the course of the run. It is strongly recommended that you redirect the output to a file, rather than simply streaming it to the screen, as shown in the script in Fig. A.1. To monitor the run, use the “tail” command, as seen in Fig. A.2. As you can see, once the code starts up, the screen output is quite minimal. To increase the amount of output, use the “<debugging>” variable in the main case setup. The minimal output consists of the timestep size and its change from the last timestep, the number of the timestep, and notices such as “Restart file written” or “Integrated force calculated.”

A.5.1 Forcing a restart file output

To force IBSEN to write a restart file after the current timestep is complete (even when the current timestep would not normally include an output), create a file called “WRITE” in the current run directory. The file does not need to contain anything, and will be deleted when the write is complete. The easiest way to do this is to use the “touch” command:

prompt\$> touch WRITE
A.5.2 Stopping IBSEN early

To stop IBSEN before it has completed the requested number of timesteps, create a file called “STOP” in the current run directory. This will tell IBSEN to stop when the current timestep is complete. Most of the time you will also want IBSEN to write out a restart file: to do this, create both “WRITE” and “STOP” files:

```
prompt\$> touch WRITE STOP
```

IBSEN will write the current timestep and then exit, printing out the usual run diagnostic information.

A.6 Visualizing the Output

During the course of a normal run, IBSEN writes out a number of different files, depending on your input file settings. Most of these files can be visualized with a number of different packages, including Paraview, Tecplot™, Fieldview™, or Microsoft Excel™.

A.6.1 3D Flowfield Data

To allow flowfield visualization with Fieldview and Tecplot (and to allow restarting in the middle of a case) IBSEN writes out Plot3D-formatted data files. These are written in double-precision, Fortran UNFORMATTED, multi-zone, Plot3D whole format. They can be visualized easily in Either Fieldview or Tecplot, though Fieldview only supports a limited subset of the information stored in the IBLANK variable. To use the IBLANKING information the same way in Tecplot, open the “Value Blanking” dialog box and create a filter based in
Table A.1: IBLANK Values for Tecplot Visualization

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
<th>Data Valid?</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Exterior body edge (BC applied)</td>
<td>Yes</td>
</tr>
<tr>
<td>1</td>
<td>Standard field point</td>
<td>Yes</td>
</tr>
<tr>
<td>0</td>
<td>Receptor, data set by oversetting</td>
<td>Yes</td>
</tr>
<tr>
<td>-1</td>
<td>Superseded by a higher-priority overset grid</td>
<td>No</td>
</tr>
<tr>
<td>-2</td>
<td>Edge point interior to the body (no BC applied)</td>
<td>No</td>
</tr>
<tr>
<td>-3</td>
<td>Interior non-edge point</td>
<td>No</td>
</tr>
<tr>
<td>-4</td>
<td>Disabled exterior edge</td>
<td>No</td>
</tr>
</tbody>
</table>

the IBLANK variable, blanking out all cells with all of the nodes less than zero. In Tecplot the IBLANK array can also be visualized directly to see the status of the oversetting as well as the body edge detection. Table A.1 lists the various values of the IBLANK variable.

If a separate visualization write controller is enabled, VTK-formatted visualization files are written out for visualization in Paraview (or any other VTK-based visualization package).

### A.6.2 Surface Pressure Distribution Files

Surface pressure distributions are written to a Tecplot-formatted data file by the dedicated surface pressure write controller, and to a VTK-formatted data file by the visualization write controller.

### A.6.3 Integrated Body Forces Files

Integrated body forces are written to a simple columnar data file in CSV format for visualization in Excel, Tecplot, Paraview, etc.
A.6.4 Triangulated Surface Data Files

The visualization write controller also writes out VTK-formatted surface definition files with a large amount of geometry analysis data.

A.6.5 Boundary Condition Error Files

If there is an error with your boundary conditions an error message will be written directing you to a file that contains the points where the boundary conditions were not specified. This file can be loaded into Tecplot.

A.7 Interfacing with PSU-WOPWOP

When using an Acoustic Data surface (see Section A.4.8) IBSEN generates PSU-WOPWOP-formatted files of the requested types. No data conversion is necessary to use with current versions of PSU-WOPWOP. These files are all time-dependent, and already contain whatever motions were specified for the acoustic data surface. For example, if the acoustic data surface is a compact patch moving along with the rotor blades, the output file already contains all of the rotor blade motions. The only motion missing is a forward flight speed, since it is typical to run IBSEN with an oncoming freestream velocity rather than actually moving the geometries forward (i.e. more like a wind tunnel than a flight test). In this case, only a single motion needs to be specified for PSU-WOPWOP: the forward flight speed (set to the negative of the IBSEN freestream velocity). All of the other motions are already encapsulated in the patch file(s). If periodic outputs were requested, PSU-WOPWOP will treat the data files as periodic, even if the signal is not; care must be taken to ensure that the flow solver has converged.
to a steady-state solution (in the rotating frame, of course) before using periodic inputs to PSU-WOPWOP. Any discontinuity between the endpoints will appear as potentially very-high-amplitude jumps in the flow data. In general it is best to run with aperiodic input data to avoid this situation.
Bibliography


Vita
Christopher C. Hennes

Norman Public Library
Pioneer Library System
Norman, OK 73072
Office: (405) 701-2697

Education

September 2005 to December 2013
Ph.D. Aerospace Engineering
The Pennsylvania State University
Chair: Dr. Kenneth S. Brentner

September 2003 to August 2005
M.S. Aerospace Engineering
The Pennsylvania State University
Thesis: The Effect of Blade Flexibility on Rotorcraft Acoustics
Advisor: Dr. Kenneth S. Brentner

September 1998 to December 2003
B.S. Aerospace Engineering
Iowa State University

Employment History

CTC Assistant, Pioneer Library System (Norman, OK)
September 2012 – Present

Consultant, Vortex Consulting LLC (Norman, OK)
May 2009 – Present

Graduate Research Assistant, Penn State University (University Park, PA)
June 2003 – May 2010

Aerospace Engineer, Sukra Helitek (Ames, IA)
January 2000 – June 2003