Potential Flow Modelling for Wind Turbines

by

Shane Cline
B.Sc., University of Toledo, 2003
M.Sc., Purdue University, 2005

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Mechanical Engineering

© Shane Cline, 2011
University of Victoria

All rights reserved. This thesis may not be reproduced in whole or in part, by
photocopying or other means, without the permission of the author.
Potential Flow Modelling for Wind Turbines

by

Shane Cline
B.Sc., University of Toledo, 2003
M.Sc., Purdue University, 2005

Supervisory Committee

Dr. C. Crawford, Supervisor
(Department of Mechanical Engineering)

Dr. B. Buckham, Department Member
(Department of Mechanical Engineering)
Supervisory Committee

Dr. C. Crawford, Supervisor
(Department of Mechanical Engineering)

Dr. B. Buckham, Department Member
(Department of Mechanical Engineering)

ABSTRACT

Lagrangian potential flow methods are a promising alternative to mainstream wind turbine aerodynamics tools such as blade element momentum methods and grid-based computation fluid dynamics approaches. Potential flow methods are relatively easy to setup and robust with respect to geometry. With the advent of numerical techniques such as the fast multipole method, potential flow methods can be made computationally fast. Viscous core modelling has led to improvements in accuracy and numerical robustness. A C++ programming library employing Prandtl-Weissinger lifting line wing models and tailorble potential flow wake models has been developed under the name LibAero. The library offers steady-state, periodic, and unsteady flow simulators that can be used interchangeably with wake models. (Periodic and unsteady simulation are still under development and validation.) Wake models are constructed from potential flow elements such as vortex particles, filaments, and sheets. Fast multipole method, symmetry modelling, multigrid method, and relaxation iteration are utilized to accelerate the computation of element-by-element interactions. The computational performance is assessed and the numerical results are validated against wind tunnel experimental data from the MEXICO Project and the Tjæreborg wind turbine. The results of steady-state simulations with respect to a variety of numerical options and rotor blade designs are presented and conclusions are drawn.
# Contents

Supervisory Committee ii  
Abstract iii  
Table of Contents iv  
List of Tables viii  
List of Figures ix  
Nomenclature xii  
Acknowledgements xv  
Dedication xvi  

## 1 Introduction 1  
1.1 Motivation ................................................. 1  
1.2 Background .................................................. 2  
1.2.1 Overview of Low-Speed Aerodynamics ................. 2  
1.2.2 Object-Oriented Software Design ..................... 4  
1.3 Literature Review .......................................... 7  
1.3.1 Early Research .......................................... 7  
1.3.2 Recent Developments ................................... 9  
1.3.3 Fast Multipole Method ................................. 10  
1.4 Key Contributions ......................................... 11  
1.5 Coordinate System ........................................ 13  
1.6 Thesis Outline ............................................. 13  

## 2 Potential Flow Theory 15
## 2.1 Governing Equations

- **2.1.1 Eulerian Perspective**
- **2.1.2 Lagrangian Perspective**
- **2.1.3 Boundary Conditions**

## 2.2 Potential Flow Elements

- **2.2.1 Vortex Particle**
- **2.2.2 Vortex Filament**
- **2.2.3 Vortex Quad**
- **2.2.4 Source Particle**
- **2.2.5 Doublet Particle**
- **2.2.6 Free Stream**

## 2.3 Vortex Core Models

- **2.3.1 Gaussian Vortex Particle Core**
- **2.3.2 Garrel Vortex Filament Core**
- **2.3.3 Vatistas Family of Core Models**

## 2.4 Prandtl-Weissinger Lifting Line

## 2.5 Tree-Code Algorithms

## 2.6 Relaxation Iteration

## 3 Simulation Architecture

### 3.1 Influence

### 3.2 Aerodynamic Data Classes

- **3.2.1 AeroElement**
- **3.2.2 AeroNode**
- **3.2.3 AeroAssembly**
- **3.2.4 AeroState**

### 3.3 Aerodynamic Algorithm Classes

- **3.3.1 FreeStream**
- **3.3.2 Kernel**
- **3.3.3 AeroEquation**

### 3.4 Wing Data Classes

- **3.4.1 Wing**
- **3.4.2 WingStation**
- **3.4.3 WingNode**
- **3.4.4 WingAssembly**
3.4.5 WingState .................................................. 59
3.5 Wing Algorithm Classes ........................................ 59
  3.5.1 WingEquation ............................................. 59
3.6 Programming Pitfalls ........................................... 62
  3.6.1 Separation of data and algorithms ......................... 62
  3.6.2 Shared Nodes .............................................. 63
  3.6.3 Deep Copying .............................................. 64
  3.6.4 Nested Object Arrays .................................... 65
3.7 Simulation Execution ........................................... 66

4 Performance Evaluation ........................................... 73
  4.1 Numerical Diagnostics ........................................ 74
    4.1.1 Effects of Wake Truncation ............................... 74
    4.1.2 Time Step Discretization ............................... 77
    4.1.3 Blade Section Discretization ............................ 79
    4.1.4 Convergence Behaviour .................................. 80
  4.2 Computational Acceleration .................................. 81
    4.2.1 Relaxation Method ...................................... 82
    4.2.2 Barnes-Hut Tree Code .................................. 84
    4.2.3 Fast Multipole Method .................................. 85
  4.3 Induction and Evolution Equations ........................... 87
    4.3.1 Viscous Core Modelling .................................. 88
    4.3.2 Advection Models ...................................... 89
    4.3.3 Diffusion Models ...................................... 90
  4.4 Wake Models and Settings ................................... 92
    4.4.1 Wake Models ........................................... 92
    4.4.2 Vortex Particle Transition and Spacing ................ 94
  4.5 Experimental Validation .................................... 95
    4.5.1 MEXICO Experiment .................................... 96
    4.5.2 Tjæreborg Wind Turbine ................................. 107

5 Applications ..................................................... 109
  5.1 Steady Axial Flow Simulations ............................... 110
    5.1.1 Coned Wind Rotor ...................................... 110
    5.1.2 Swept-Bladed Wind Rotor ............................... 113
5.1.3 Wind Turbine Rotor with Winglets ........................................ 117
5.2 Periodic Flow Simulations ...................................................... 120

<table>
<thead>
<tr>
<th>6 Conclusions and Future Work</th>
<th>122</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Conclusions</td>
<td>122</td>
</tr>
<tr>
<td>6.2 Future Work</td>
<td>125</td>
</tr>
</tbody>
</table>

Bibliography ......................................................... 127
List of Tables

Table 3.1 Pitfalls and achievements of LibAero development ............ 66

Table 4.1 LibAero simulation settings ........................................ 73
Table 4.2 Effect of truncation .................................................... 77
Table 4.3 Effect of temporal resolution ........................................ 79
Table 4.4 Effect of bladed discretization ..................................... 79
Table 4.5 Effect of iteration number .......................................... 81
Table 4.6 Effect of relaxation setting ......................................... 84
Table 4.7 Effect of Barnes-Hut tree-code (BHTC) cutoff setting ....... 85
Table 4.8 Effect of fast multipole method (FMM) cutoff setting ....... 86
Table 4.9 Effect of element choice on FMM ................................. 87
Table 4.10 Effect of viscous core models ................................. 89
Table 4.11 Effect of advection models .................................. 90
Table 4.12 Effect of turbulent viscosity .................................. 91
Table 4.13 Effect of wake models ................................................. 93
Table 4.14 Effect of vortex particle spacing ............................... 95
Table 4.15 ExcelBEM simulation settings ............................... 97
Table 4.16 MEXICO power and thrust results comparison ............ 100
Table 4.17 MEXICO velocimetry vs. LibAero flow field ............... 107

Table 5.1 Default simulation settings ........................................... 110
Table 5.2 Effect of coning angle ............................................. 113
Table 5.3 Effect of blade sweep .............................................. 116
Table 5.4 Effect of winglet toe angle ...................................... 118
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Weissinger lifting line and wake</td>
<td>8</td>
</tr>
<tr>
<td>1.2</td>
<td>Quad-tree</td>
<td>11</td>
</tr>
<tr>
<td>1.3</td>
<td>Coordinate system from downwind perspective</td>
<td>13</td>
</tr>
<tr>
<td>2.1</td>
<td>Diagram of rotation, advection, diffusion, and deformation</td>
<td>21</td>
</tr>
<tr>
<td>2.2</td>
<td>Potential flow elements</td>
<td>30</td>
</tr>
<tr>
<td>2.3</td>
<td>Vortex filament</td>
<td>32</td>
</tr>
<tr>
<td>2.4</td>
<td>Vortex quad discretization</td>
<td>34</td>
</tr>
<tr>
<td>2.5</td>
<td>Superposition of Gaussian core distributions</td>
<td>35</td>
</tr>
<tr>
<td>2.6</td>
<td>N-body binary tree</td>
<td>42</td>
</tr>
<tr>
<td>3.1</td>
<td><em>Influence</em> class hierarchy</td>
<td>47</td>
</tr>
<tr>
<td>3.2</td>
<td><em>AeroElement</em> class hierarchy</td>
<td>48</td>
</tr>
<tr>
<td>3.3</td>
<td><em>AeroNode</em> class hierarchy</td>
<td>49</td>
</tr>
<tr>
<td>3.4</td>
<td><em>AeroAssembly</em> class hierarchy</td>
<td>50</td>
</tr>
<tr>
<td>3.5</td>
<td><em>AeroState</em> class hierarchy</td>
<td>51</td>
</tr>
<tr>
<td>3.6</td>
<td><em>FreeStream</em> class hierarchy</td>
<td>51</td>
</tr>
<tr>
<td>3.7</td>
<td><em>Kernel&lt;AeroElement&gt;</em> class hierarchy</td>
<td>52</td>
</tr>
<tr>
<td>3.8</td>
<td><em>Kernel&lt;AeroAssembly&gt;</em> class hierarchy</td>
<td>52</td>
</tr>
<tr>
<td>3.9</td>
<td><em>Kernel&lt;T&gt;</em> class hierarchy</td>
<td>53</td>
</tr>
<tr>
<td>3.10</td>
<td><em>AeroEquation&lt;AeroNode&gt;</em> class hierarchy</td>
<td>55</td>
</tr>
<tr>
<td>3.11</td>
<td><em>WingObject</em> class hierarchy</td>
<td>57</td>
</tr>
<tr>
<td>3.12</td>
<td><em>WingEquation&lt;WingNode&gt;</em> class hierarchy</td>
<td>59</td>
</tr>
<tr>
<td>3.13</td>
<td>Blade section velocities and forces</td>
<td>61</td>
</tr>
<tr>
<td>3.14</td>
<td>Overall program flow chart</td>
<td>67</td>
</tr>
<tr>
<td>3.15</td>
<td>Simulation flow chart</td>
<td>69</td>
</tr>
<tr>
<td>3.16</td>
<td>Order-of-update for lifting line and wakes</td>
<td>70</td>
</tr>
<tr>
<td>3.17</td>
<td>Example rotor wake</td>
<td>71</td>
</tr>
</tbody>
</table>
Figure 4.1 Depiction of truncation length .......................................... 75
Figure 4.2 Effect of truncation on angles-of-attack ............................. 76
Figure 4.3 Effect of truncation on flow induction ............................... 77
Figure 4.4 Depiction of temporal resolution ...................................... 78
Figure 4.5 Effect of temporal resolution on angles-of-attack ............... 78
Figure 4.6 Effect of blade discretization on angles-of-attack ............... 80
Figure 4.7 Effect of iteration number on angles-of-attack .................... 81
Figure 4.8 Effect of temporal resolution on convergence .................... 83
Figure 4.9 Effect of relaxation setting on angles-of-attack ................... 83
Figure 4.10 Effect of BHTC cutoff setting on angles-of-attack ............. 85
Figure 4.11 Effect of FMM cutoff setting on angles-of-attack ............... 86
Figure 4.12 Effect of viscous core models on angles-of-attack ............. 88
Figure 4.13 Effect of advection models on angles-of-attack .................. 90
Figure 4.14 Effect of turbulent viscosity on angles-of-attack ............... 91
Figure 4.15 Wake models .......................................................... 92
Figure 4.16 Effect of wake model on angles-of-attack ......................... 93
Figure 4.17 Vortex particle transition ........................................... 94
Figure 4.18 Effect of vortex particle spacing on angles-of-attack .......... 95
Figure 4.19 MexNext wind rotor configuration .................................. 97
Figure 4.20 MEXICO blade section forces ...................................... 99
Figure 4.21 MEXICO axial induction ............................................. 102
Figure 4.22 MEXICO tangential induction ....................................... 103
Figure 4.23 MEXICO angles-of-attack .......................................... 105
Figure 4.24 Comparison with particle image velocimetry results ........... 106
Figure 4.25 Tjæreborg power and thrust ........................................ 107

Figure 5.1 Coned rotors ............................................................ 111
Figure 5.2 Coned rotor loadings .................................................. 112
Figure 5.3 Swept-bladed rotors .................................................... 114
Figure 5.4 Graph of blade sweep ................................................ 115
Figure 5.5 Swept rotor loadings ................................................... 116
Figure 5.6 Angles-of-attack on swept blades ................................... 117
Figure 5.7 Two-bladed rotor with winglets ..................................... 118
Figure 5.8 Local flow on blades with winglets ................................ 119
Figure 5.9 Wingleted rotor loadings ............................................. 120
Figure 5.10  Snapshot of a steady yaw rotor wake  . . . . . . . . . . . . . .  121
Nomenclature

Vector and tensor variables are represented as bold letters while scalar variables are represented with plain font.

Greek symbols

\begin{itemize}
  \item \( \alpha \) Angle-of-attack \hspace{1cm} Degrees
  \item \( \alpha \) Vortex particle strength \hspace{1cm} m\(^3\)/s
  \item \( \Gamma \) Circulation \hspace{1cm} m\(^2\)/s
  \item \( \eta \) Perpendicular distance \hspace{1cm} m
  \item \( \theta \) Angle \hspace{1cm} Degrees
  \item \( \kappa \) Core distribution \hspace{1cm} m\(^{-3}\)
  \item \( \lambda \) Diffusivity \hspace{1cm} m\(^2\)/s
  \item \( \mu \) Mean \hspace{1cm} (variable)
  \item \( \mu \) Blade spanwise coordinate \hspace{1cm} (dimensionless)
  \item \( \nu \) Fluid kinematic viscosity \hspace{1cm} m/s
  \item \( \nu_T \) Fluid turbulent viscosity \hspace{1cm} m\(^2\)/s
  \item \( \rho \) Fluid density \hspace{1cm} kg/m\(^3\)
  \item \( \sigma \) Source density \hspace{1cm} m\(^{-1}\)s\(^{-1}\)
  \item \( \sigma \) Gaussian core radius \hspace{1cm} m
  \item \( \tilde{\sigma} \) Non-Gaussian core radius \hspace{1cm} m
  \item \( \phi \) Scalar potential \hspace{1cm} m\(^3\)/s
  \item \( \psi \) Vector potential or stream function \hspace{1cm} m\(^3\)/s
  \item \( \omega \) Vorticity \hspace{1cm} s\(^{-1}\)
\end{itemize}

Latin symbols

\begin{itemize}
  \item \( a \) Source element strength \hspace{1cm} m\(^2\)/s
  \item \( \mathbf{a} \) Acceleration \hspace{1cm} m\(^2\)/s
  \item \( A \) Airfoil cross-sectional area \hspace{1cm} m\(^2\)
  \item \( A \) Standard radius cutoff multiplier \hspace{1cm} (dimensionless)
  \item \( A \) First element node \hspace{1cm} (subscript)
  \item \( AR \) Aspect ratio \hspace{1cm} (dimensionless)
\end{itemize}
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>Doublet strength</td>
<td>$m^3/s$</td>
</tr>
<tr>
<td>$B$</td>
<td>Second element node</td>
<td>(subscript)</td>
</tr>
<tr>
<td>$c$</td>
<td>Chord vector</td>
<td>$m$</td>
</tr>
<tr>
<td>$c$</td>
<td>Element centre</td>
<td>$m$</td>
</tr>
<tr>
<td>$c$</td>
<td>Chord</td>
<td>$m$</td>
</tr>
<tr>
<td>$C$</td>
<td>Third element node</td>
<td>(subscript)</td>
</tr>
<tr>
<td>$C$</td>
<td>Node centre</td>
<td>$m$</td>
</tr>
<tr>
<td>$C_L$</td>
<td>Lift coefficient</td>
<td>(dimensionless)</td>
</tr>
<tr>
<td>$C_D$</td>
<td>Drag coefficient</td>
<td>(dimensionless)</td>
</tr>
<tr>
<td>$C_M$</td>
<td>Pitching moment coefficient</td>
<td>(dimensionless)</td>
</tr>
<tr>
<td>$d$</td>
<td>Element bounding sphere diameter</td>
<td>$m$</td>
</tr>
<tr>
<td>$D$</td>
<td>Node bounding sphere diameter</td>
<td>$m$</td>
</tr>
<tr>
<td>$D$</td>
<td>Fourth element node</td>
<td>(dimensionless)</td>
</tr>
<tr>
<td>$D$</td>
<td>Drag force</td>
<td>$kg \cdot m/s^2$</td>
</tr>
<tr>
<td>$f$</td>
<td>Conservative field force</td>
<td>$kg/m^2s^2$</td>
</tr>
<tr>
<td>$F$</td>
<td>Resultant force</td>
<td>$kg \cdot m/s^2$</td>
</tr>
<tr>
<td>$G$</td>
<td>Velocity gradient function</td>
<td>$s^{-1}$</td>
</tr>
<tr>
<td>$G$</td>
<td>Gaussian distribution</td>
<td>$m^{-1}$</td>
</tr>
<tr>
<td>$I$</td>
<td>Induced influence function</td>
<td>$m/s$ or $s^{-1}$</td>
</tr>
<tr>
<td>$I$</td>
<td>Identity matrix</td>
<td>(dimensionless)</td>
</tr>
<tr>
<td>$k$</td>
<td>General integer index variable</td>
<td>(subscript)</td>
</tr>
<tr>
<td>$k$</td>
<td>Blade section thickness vector</td>
<td>$m$</td>
</tr>
<tr>
<td>$K$</td>
<td>Core model multiplier</td>
<td>(dimensionless)</td>
</tr>
<tr>
<td>$L$</td>
<td>Lift force</td>
<td>$kg \cdot m/s^2$</td>
</tr>
<tr>
<td>$M$</td>
<td>Pitching moment</td>
<td>$kg \cdot m^2/s^2$</td>
</tr>
<tr>
<td>$n$</td>
<td>Exponent parameter</td>
<td>(dimensionless)</td>
</tr>
<tr>
<td>$n$</td>
<td>Unit normal vector</td>
<td>(dimensionless)</td>
</tr>
<tr>
<td>$N$</td>
<td>Integer parameter</td>
<td>(subscript)</td>
</tr>
<tr>
<td>$p$</td>
<td>Pressure</td>
<td>$kg/ms^2$</td>
</tr>
<tr>
<td>$Pr_T$</td>
<td>Turbulent Prandtl number</td>
<td>(dimensionless)</td>
</tr>
<tr>
<td>$q$</td>
<td>Generic field quantity</td>
<td>(variable)</td>
</tr>
<tr>
<td>$Q$</td>
<td>Total quantity</td>
<td>(variable)</td>
</tr>
<tr>
<td>$r$</td>
<td>Displacement</td>
<td>$m$</td>
</tr>
<tr>
<td>$R$</td>
<td>Distance, $</td>
<td>r</td>
</tr>
<tr>
<td>$Re$</td>
<td>Reynolds number</td>
<td>(dimensionless)</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
<td>Unit</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>$s$</td>
<td>Curvilinear path</td>
<td>$m$</td>
</tr>
<tr>
<td>$s$</td>
<td>Element vortex monopole strength</td>
<td>$m^3/s$</td>
</tr>
<tr>
<td>$S$</td>
<td>Node vortex monopole strength</td>
<td>$m^3/s$</td>
</tr>
<tr>
<td>$Sc$</td>
<td>Schmidt number</td>
<td>(dimensionless)</td>
</tr>
<tr>
<td>$t$</td>
<td>Time</td>
<td>$s$</td>
</tr>
<tr>
<td>$T$</td>
<td>Fluid deviatoric stress tensor</td>
<td>$kg/ms^2$</td>
</tr>
<tr>
<td>$Tr$</td>
<td>Thickness ratio</td>
<td>(dimensionless)</td>
</tr>
<tr>
<td>$u$</td>
<td>Flow velocity</td>
<td>$m/s$</td>
</tr>
<tr>
<td>$U$</td>
<td>Velocity function</td>
<td>$m/s$</td>
</tr>
<tr>
<td>$v$</td>
<td>Solid velocity</td>
<td>$m/s$</td>
</tr>
<tr>
<td>$V$</td>
<td>Volume</td>
<td>$m^3$</td>
</tr>
<tr>
<td>$V$</td>
<td>Variance</td>
<td>(variable)</td>
</tr>
<tr>
<td>$x$</td>
<td>Cartesian coordinate</td>
<td>$m$</td>
</tr>
<tr>
<td>$x$</td>
<td>Position</td>
<td>$m$</td>
</tr>
<tr>
<td>$y$</td>
<td>Cartesian coordinate</td>
<td>$m$</td>
</tr>
<tr>
<td>$z$</td>
<td>Cartesian coordinate</td>
<td>$m$</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

I would like to thank:

my family, for supporting me in the low moments.

Curran Crawford, for intellectual guidance, encouragement, and above all patience.

Pacific Century Scholarship and University of Victoria Fellowship for funding myself and other Sustainable Systems Design Laboratory colleagues with scholarship and research support.

Ever since the beginning of modern science, the best minds have recognized that the range of acknowledged ignorance will grow with the advance of science.

Unfortunately, the popular effect of this scientific advance has been a belief, seemingly shared by many scientists, that the range of our ignorance is steadily diminishing and that we can therefore aim at more comprehensive and deliberate control of all human activities. It is for this reason that those intoxicated by the advance of knowledge so often become the enemies of freedom.

Friedrich August von Hayek
DEDICATION

To my family: To my wife, Jinfei, for her moral support, financial support, and most importantly culinary expertise.

To my mother, father, and brother for encouragement over the years.

To my friends and colleagues in the Sustainable System Design Laboratory for profound intellectual discussions of renewable energies topics and political economy.
Chapter 1

Introduction

Horizontal-axis wind turbine (HAWT) aerodynamics analysis has three major approaches: blade element momentum (BEM), potential flow (PF), and grid-based computational fluid dynamics (CFD). Each approach offers advantages as well as disadvantages when compared to the others [1]. The choice of method depends on the objectives of the practitioner.

Currently, BEM methods are the industry standard for wind turbine aerodynamic analyses. BEM provides low-fidelity aerodynamics simulation, which finds its niche in preliminary and conceptual design stages where computational speed and easy setup are required [2]. Full-field CFD modelling plays an important role in design validation and time-domain certification where high-fidelity is required, oftentimes for modelling extreme events [3]. PF methods are less popular in wind turbine engineering, but are receiving attention from researchers [4]. The mid-fidelity approach of the PF methodology offers a compromise between BEM and CFD approaches. This work is based on the development of a general-purpose programming library for PF aerodynamics simulations under the name LibAero, and its application to horizontal-axis wind turbine modelling.

1.1 Motivation

The long-term aim of this line of research is to apply multidisciplinary design optimization (MDO) methodologies to the design of conventional and unconventional wind turbines. The MDO approach raises the possibility of exploring a wide array of designs in an efficient manner by delegating more of the exploratory tactics to the computer,
thus allowing the practitioner to focus on aspects where humans excel such as numerical strategy and design feasibility. The goal is to produce an efficient process for arriving at optimal wind turbine designs and efficient design families. For instance, a family of upwind HAWT rotors with raked winglets might be found as an efficient design family; in that case further optimization of the raking angle could be carried out for a specified wind flow condition. This work aims to provide only the aerodynamics module, which would serve an integral role in the proposed MDO framework. To quickly explore a wide variety of designs and design families, the aerodynamics simulation component should satisfy two criteria:

1. *Computational speediness* - Preliminary and conceptual design stages involve a great number of individual simulation runs. CFD modelling is not advisable because the computational and setup expense per simulation run is prohibitive. PF methods, although more computationally intensive than BEM, present a promising alternative for design purposes [5].

2. *Wide range of validity* - Validity across all possible design iterates allows a large design space to be explored. BEM methods rely on actuator disc momentum modelling and employ tip-corrections to account for the finite blade number. Semi-empirical techniques that are inherent in BEM impede the exploration of novel design spaces such as coned and wingleted rotors, as well as multi-rotor systems. PF methods provide ample flexibility in this regard.

### 1.2 Background

#### 1.2.1 Overview of Low-Speed Aerodynamics

Low-speed wind turbine aerodynamics can be divided into three categories according to the technique of solving for flows and aerodynamic forces. These techniques are BEM, PF, and CFD, which are described in the following paragraphs.

1. *Blade element momentum theory* - This category includes BEM for rotor discs as well as the stream-tube models used in vertical-axis wind turbine (VAWT) modelling. Beneath the façade, BEM-type models are simplified PF methods that are adjusted with global momentum balances and other semi-empirical corrections. For example, the BEM flow field is equivalent to a wake model of
concentric vortex cylinders [3] where the strength of each cylinder depends on airfoil coefficients, thus accounting for radial variation in the flow. BEM methods employ a momentum balance based on Bernoulli’s equation, which typically includes Glauert’s momentum correction to adjust for turbulent mixing in the wake region. Prandtl’s tip loss correction accounts for the effect that finite number of blades has on the flow field, thus partially including azimuthal variation. Together, these effects iteratively determine the strength and configuration of the vortex cylinders, thus determining the flow field and blade forces.

In BEM, the computational problem is reduced to one physical dimension; all quantities are functions of radial position on the rotor disc. In particular, axial induction factor and tangential induction factor are the two quantities that are iteratively solved at each blade station. This level of simplicity yields fast computing but reduces accuracy and range of validity [2]. From experience with the ExcelBEM code by Crawford [5], the typical computational run times are in the range of a few seconds to a few minutes, which likely applies to commercial BEM codes such as Bladed\textsuperscript{TM} [6] and AeroDyn\textsuperscript{TM} [7, 8, 9].

2. *Lagrangian potential flow methods* - This category includes vortex methods that rely on Lagrangian vorticity tracking as well as vortex-panel methods where the solid-body panels are composed of source and doublet elements. Vortex, source, and doublet elements are types of PF elements since their induced flow equations are harmonic functions. This means the induced flow of PF elements satisfies Laplace’s equation except at singularity points and therefore upholds mass conservation [4]. Furthermore, popular vortex elements have simple geometries such as points, lines, planes, line segments, rings, and cylinders.

The central concept of PF methods is the superposition of numerous PF elements to form a numerical description of the entire flow field. The element-based approach allows resolution to be tailored, for instance changing discretization parameters [4]. Such flexibility obviates the need for most types of empirical corrections. The code presented in this work relies on experimental airfoil data and corrections to account for dynamic stall phenomena, just as BEM does. The Lagrangian approach of mainstream PF methods allows PF elements to systematically position themselves in the flow field where vorticity is greatest. As such the computational problem is reduced to a two-dimensional region corresponding to the wake sheet that trails behind each wing or blade. Experience
with *LibAero* indicates that typical computational run times of PF codes are in the range of a few minutes to a few hours.

3. *Eulerian Meshed CFD* - This category includes the finite difference method (FDM), finite volume method (FVM), and finite element method (FEM) flow solvers. Flow solvers of this type are generally Eulerian, and include coupled mass, momentum, and energy equations. In other words, the mesh does not advect with any flow quantity. To account for rotation of the rotor, the CFD mesh typically requires a special disc region surrounding the rotor that follows a rotating reference frame [1]. Beyond turbulence modelling, empirical corrections are mostly unnecessary. A great number of elements are required to model the fluid boundary layer in vicinity of the wings. Alternatively, tabulated airfoil coefficients can be employed, sacrificing generality and recovering some computational speed [1]. Excess elements are required to fill the three-dimensional space as well, even in areas where gradient values such as vorticity are small. Along with this abundance of elements comes great computational expense.

To partially ameliorate the computational expense, mesh refinement techniques are able to enhance the numerical resolution in high vorticity regions. Models produced by Schmitz [10] and Stone [11], employ a CFD solver for solid surfaces such as wings, and interface to a PF solver for the wake region. Stone [11] argues that the coupled method provides a better representation of the wake region than pure CFD due to numerical dissipation of vorticity, which is inherent to meshed solutions of hyperbolic partial differential equations (PDEs). Experience with CFD for actuator line modelling has shown that typical meshed-CFD run times are in the hour to day range.

### 1.2.2 Object-Oriented Software Design

In order to author an efficient, modular, clear, and concise simulation code base, the first design decision was to choose a programming paradigm that would confer an effective organizational strategy and facilitate creative problem solving. Programming paradigms that are frequently utilized in numerical coding are procedural programming, object-oriented programming (OOP), and functional programming. Among these the functional paradigm is newer, and thus not available in the high performance languages such as C++. Having selected C++ over MATLAB\textsuperscript{TM}, C, and Fortran for
performance and maintainability reasons, both procedural and OOP approaches are available, as well as any self-consistent combination of the two approaches.

The simplest programming paradigm is procedural programming. In procedural programming there is a clear distinction between data and functions. Functions receive data as arguments and operate on the data. Additionally, functions can create new chunks of data during run time. This degree of simplicity has both advantages and disadvantages. The major advantage is the low-level control the programmer has when programming for special hardware and high-performance applications. Another advantage is the intuitive programming style of writing sequential commands. The disadvantage is the difficulty and time spent programming with a procedural language that lacks user-friendly software design patterns.

The OOP paradigm, while not having additional computational capabilities beyond procedural programming, provides user-friendly language features that make software development easier to accomplish. The main feature that distinguishes OOP is the design pattern known as polymorphism. Polymorphism, in the programming context, is the ability to use a specialized data type in place of a more general data type to carry out a specific task. For example, dog is a specialized type of animal. If an animal can speak, dogs can be set to bark when asked to speak. Class, object, inheritance, virtual method, dynamic dispatch, and interface are words that are used to define concepts related to OOP and polymorphism [12].

In a nutshell, OOP has a language feature called a class. A class is a data type bundled with functions called methods. As a data type, a class stores certain kinds of information. For instance, a class that contains three numbers could represent x, y, and z coordinates. Suitable names for this class would be CartesianTriple, XYZ, or Vector3. A particular instance of a class is called an object. For example, the command XYZ $a = XYZ(3,4,5)$ implies that $a$ is an object of the $XYZ$ class, and that $a$ has a value of $XYZ(3,4,5)$.

The group of methods that belong to the same class are called an interface. Class methods are given automatic access to class data. Like a function, a method receives arguments, but the OOP syntax differs. In procedural programming, the dot product of two $XYZ$ objects $a$ and $b$ is expressed as $\text{dot}(a,b)$. In OOP, the dot product can alternatively be expressed as $a\cdot b$. This is akin to word order in spoken languages such as Subject-Verb-Object. The OOP approach promotes software design according to organizational principles that humans typically apply to problem solving. Particularly, class objects are components that store large amounts of data internally.
and pass small amounts of information between one another in order to perform tasks that depend on the information received. On the flip side, some programmers feel that the OOP approach is a herding mechanism for software projects with a large number of programmers, and experience significant restrictions in problem-solving capacity when employing OOP techniques [13].

A class can inherit from another class. Inheritance means the derived (child) class contains all the data entries and interface methods of the base (parent) class. Additional data and methods can be defined inside the derived class. If a method in the base class is virtual, a method of the same name and signature composed in the derived class will override that method with a new implementation. The derived class can be used in any situation that the parent class is accepted. In such a case, the overridden methods will be called during execution. This dynamic choice of method calling at run time is referred to as dynamic dispatch [12]. For instance, dynamic dispatch allows a VortexParticle object and a VortexFilament object to provide the same interface and same overall behaviour, while calling different algorithms to induce flow velocity or evolve in the flow field. Chapter 3 explains the class hierarchy and its significance for dynamic dispatch.

Another feature that C++ provides in addition to its C language base is templates, which are sometimes called generics. Because C, C++, and FORTRAN are static-typed languages, as opposed to dynamic-typed languages such as MATLAB and Python, the programmer must specify the data type of every variable in order to assist the compiler in producing optimized machine code. Examples of primitive data types are int and float, which represent integer numbers and floating point numbers respectively. Data types that comprise combinations of primitive types are referred to as object types; the XYZ class mentioned previously is one such example. In some cases, however, an object class may contain objects of variable type. For generic container classes such as List, the compiler must know whether the data type is a List of int, a List of double, or a List of some other data type. Templating allows the programmer to compose the List class once, and implement it by specifying the contained type with angled bracket syntax. The syntax for a List of int appears as List<int>, and this syntax is employed in explanations given in Chapter 3.

Procedural programming is the approach taken by languages such as C, Pascal, and MATLAB. All three languages have modern, enhanced versions that provide the OOP paradigm while maintaining backwards compatibility with the original procedural languages. In the case of C, the object-oriented descendent is C++. C++ retains
the low-level controls and performance advantages of C, while allowing the programmer to utilize polymorphic techniques such as class inheritance and templates. Because C++ maintains the high performance of C, while providing numerous software design features, C++ was chosen as the language of *LibAero*.

Likewise, C++ overshadowed FORTRAN in software design features. Despite FORTRAN’s emphasis on numerical simulation, C++ design patterns are a better fit to the desired layout of *LibAero*. Research suggests that the numerical performance of C++ is comparable to FORTRAN, and the popularity of C++ brings more existing code and community support as well as more available and universal compilers [14, 15, 16]. The Computer Language Benchmarks Game by the Debian Project indicates that typical implementations of C++ benchmark programs outperform those of FORTRAN when matrix calculations are uncommon [17]. Since *LibAero* does not rely on matrix calculations, the argument in favour of C++ is strengthened.

### 1.3 Literature Review

#### 1.3.1 Early Research

The origins of lifting line and free vortex aerodynamics traces back to the analytical approaches of Lanchester and Prandtl in 1918 when early aerodynamicists were looking for practical quantitative techniques to analyze and design fixed-wing aircraft [4]. Their early model was the famous horseshoe arrangement of vortex filaments. This model has remained a viable approach through today, and it is one of the approaches pursued in this work. Although the application in this work is wind turbine aerodynamics and the computational solution methods are more advanced than those in Lanchester’s and Prandtl’s era, the vortex-tracking approaches of Lanchester and Prandtl have proven themselves as timeless.

Figure 1.1 presents one particular variation of the Prandtl’s horseshoe model that was employed in this work. The lifting line transitions into trailing vortex filaments, which in turn transition into far-wake vortex particles. This particular wing-wake model demonstrates how several types of PF elements play a role in numerical flow modelling. A similar PF element transition was employed by Willis [18]. The vortex filaments that comprise the lifting line are bound to the aerodynamic centre-line of the wing. The aerodynamic centre is near the quarter-chord position for typical airfoils [4]. The quarter-chord position, defined as the position located one-quarter of the way
along the line segment from the leading edge to the trailing edge, was used universally in this work to represent the aerodynamic centre. The nodal positions of the elements that trail are free and advect naturally in the flow. Shed filaments, which would cross the trailing filaments nearly perpendicularly, are not included, since the emphasis of this work is steady-state simulation. Steady-state simulations bring about zero circulation in the shed direction of the wake, thus only trailing filaments are depicted in Figure 1.1.

Over the years there were contributions to lifting line theory that allowed it to be applied to various wing planforms including swept-back wings [19] and twisted wings [20]. These contributions were mostly analytic approximations to the induced flow equations that gave a computationally efficient account of downwash, angles-of-attack, and hence aerodynamic forces. One simple approximation was available in Prandtl’s original work. This was the application of the zero-penetration condition at collocation points along the three-quarters chord line of the wing [4]. The improved performance of modern computers has circumvented most approximations and geometry-specific methods by making it efficient to compute induced flow velocities directly from the numerical elements of the model, in the depicted example the numerical elements are vortex filaments and particles.

The applicability of lifting line and free vortex methods to rotorcraft wakes was known as early as Prandtl’s original work in 1918. That avenue was initially taken
by Goldstein who produced analytic models of helical vortex elements, including a helical filament and a helical sheet [21]. With computing as it was in the 1920s and 1930s, free vortex methods did not lend well to rotor simulations. For rotorcraft such as autogyros and helicopters, the flow environment is inherently unsteady in the most important modelling scenarios such as forward flight. Prandtl and Glauert focused on vortex rings and cylinders to represent the wake. The merger of vortex cylinders with momentum balancing led to BEM theory [3]. Prandtl’s tip loss correction accounted for the helical structure of the rotor-trailing wake, and Glauert’s momentum correction accounted for turbulent mixing [2]. Variations of the BEM recipe have been popular ever since.

1.3.2 Recent Developments

From the 1970s and especially recently there has been renewed interest among the helicopter and wind turbine communities in applying PF methods such as vortex-panel methods to rotor simulations [3]. More than computing power alone, computational techniques have evolved to make large vortex element simulations possible; among these is the fast multipole method (FMM) developed by Greengard and Rokhlin [22]. The following paragraphs focus on the literature in the areas of PF simulations for rotor wakes and the corresponding computational acceleration techniques. Glauert authored one of the classic texts on the application of PF theory to rotorcraft and helical wakes, which may be of interest to the more captivated reader [23]. The objective is to show how this work is built on and contrasts with previous works.

Since 2000, numerous wind turbine rotor simulations based on PF elements have appeared. Among these are Voutsinas [24], Chattot [25], and Dixon [26] [27]. The first two authors focused on HAWTs while Dixon’s papers were based on the development of a VAWT simulation framework. These works employ the spherical Gaussian vortex particle core, which removes numerical singularity points and improves accuracy by accounting for viscous core diffusion of vortex elements. One advantage of particle elements is their simple point geometry as well as their spherical core models.

Vortex filament-based rotor simulations continue to be popular. Johnson [28], Wang and Coton [29], and van Garrel [30] provided influential recent works. The simulation framework by Johnson, called CAMRAD II, is versatile, applying to both wings and rotors, while having its emphasis on helicopters. Wang and Coton introduced tower shadow effects to account for flow separation in the wake of a HAWT
tower. Johnson employs a variety of vortex filament core models including the Scully core model. Wang and Coton avoid vortex singularities by prescribing the wake geometry; the model is not a free wake model, nor does it employ vortex cores. In van Garrel's code, called AWSM, a computationally speedy vortex filament viscous core model was introduced.

Since the free PF simulation methodology is relatively nascent technology, several studies have focused on testing stability and validity. Among these are Walther who showed that the overlap of diffuse vortex cores leads to numerical instability in rotor wake modelling [31]. Li conducted a study of the rotor tip vortex and noted that for largely unseparated flows, vortex core properties have little dependence on local or global Reynolds numbers [32]. On the flip side, Ramasamy and Leishman suggest that vortex core diffusion is influenced by the local Reynolds number [3, 33].

1.3.3 Fast Multipole Method

The fast multipole method (FMM), created by Greengard and Rokhlin in 1987, is a general method that employs the tree-code data structure and algorithm to accelerate solution of the N-body problem [22]. The N-body problem involves a space filled with \( N \) bodies that exert forces on all other bodies and have force exerted on themselves by all other bodies. The induced force depends on the charge, mass, or magnetic strength of the bodies and the distance between them. Force induces motion and through time-stepping the body positions are updated. The new body positions induce a new force field, and so the process continues. FMM has applications to the gravitational dynamics of star clusters, electromagnetics, and molecular dynamics [34].

Recent works by Willis [18] and He [35] have applied FMM to free vortex particle simulations. The applications were fixed-wing aircraft and rotorcraft respectively. This work progresses one step further, treating all PF elements, regardless of geometry, as bodies of the N-body problem. Included element geometries are particles, filaments, and quadrilateral sheets. In the context of flow simulation, the induced quantity is not the component force, but instead it is the component flow velocity, and in some models it is the component flow velocity gradient as well.

Figure 1.2 presents a diagram of a quad-tree data structure. The quad-tree is a structure that would be useful for two-dimensional N-body simulations. The three-dimensional analog is called an octree. The quad-tree given in Figure 1.2 demonstrates
how the tree data structures are utilized to sort bodies according to their locations. The left-hand side indicates the actual spatial arrangement. The right-hand side depicts the hierarchical structure of the tree that represents the arrangement on the left. The speed-enhancing feature of the tree-code algorithm is that each node, whether it is a branch or leaf node, contains summary information about contained bodies including some approximations. Instead of using every element directly to make calculations, the algorithm may employ the lumped nodal data when speed is preferred over accuracy. Section 2.5 provides further details of tree-code methods.

1.4 Key Contributions

The work described in this thesis combines many qualities mentioned in the previous works that were described. A wide choice of PF elements and viscous core models are available. PF elements include vortex particles, filaments, and sheets. Source and doublet elements are also available, but not used currently in any wake models. Like Johnson’s CAMRAD II (1995), LibAero is widely applicable to fixed-wings and a variety of rotor designs. Two tree-code algorithms, Barnes-Hut tree-code (BHTC) and fast multipole method (FMM), are employed for acceleration of computational dynamics. In particular, this work has begun to utilize a binary spatial partitioning (BSP) instead of an octree.
With the eventual aim to develop an MDO framework for wind turbines, suitable models of the core disciplines are required. This includes aerodynamics, structures, controls, as well as some peripheral areas such as acoustics. Among these, aerodynamics is a critical and difficult component to model. Aerodynamics is a critical component since the main purpose of a wind turbine is to extract energy from the wind. Predicting power production requires a detailed characterization of the aerodynamic forces on the rotor blades. The structures and controls portions derive their roles from the aerodynamic forces, making them in some sense secondary to aerodynamics. The aerodynamics component is a numerical coding challenge because it is typically the most computationally expensive module, and thus aerodynamics is usually the simulation bottleneck.

The objective of the current research is to produce a complete programming library (LibAero) for mid-fidelity low-speed aerodynamics simulations. The target application of the library is the design of low-speed aerodynamic machines, particularly via the MDO methodology. The intended design space includes but is not limited to standard Danish HAWTs, coned HAWTs, swept-bladed HAWTs, multi-rotor HAWTs, VAWTs, various types of tidal turbines, and ducted turbines. Fixed-wing aircraft and rotorcraft are also considered, but are of secondary importance. The objective of this thesis is to validate and demonstrate the capabilities of LibAero as well as present the theoretical and software foundations on which it is built. Moreover, the flexible programming structure of LibAero is specifically designed to facilitate experimentation with various PF formulations to ascertain the best compromise between accuracy and speed for future design work.

The scope of the thesis limits the analysis to steady-state simulation cases, however LibAero is sufficiently flexible to apply the same underlying framework to periodic and unsteady simulations. In order to maintain this flexibility, the steady-state rotor simulation requires an implicit time step that allows a finite number of finitely sized PF elements to be employed. Of course, the exact solution is attained when the time step approaches zero, however the number of PF elements required to populate the wake goes to infinity, and the physical size of each element tends to zero. Therefore, a feasible time step is defined, for instance using the duration that corresponds to a 15° angular displacement of the wind turbine rotor at its operating speed.
1.5 Coordinate System

The three-dimensional Cartesian coordinate system is employed in this thesis. In order to be clear and consistent, one particular geometric configuration was used throughout. The positive x-axis is in the direction of the HAWT rotor axis and points downstream. Negative x is upstream and positive x is downstream. The primary blade is placed on the positive y-axis with the hub at the origin. The other blades wrap around in the y-z-plane where angle $\theta$ is measured starting at 0 along the y-axis and moving toward $\pi/2$ along the z-axis. The z-axis is vertical. Figure 1.3 depicts the described coordinate system from the downwind perspective.

![Coordinate system from downwind perspective](image)

Figure 1.3: Coordinate system from downwind perspective

1.6 Thesis Outline

This thesis comprises six chapters. Chapter 1 is an introduction, which includes the motivation and the objectives of this work. Background knowledge of low-speed
aerodynamics and object-oriented software design is conveyed. The coordinate system and default orientations used throughout this document are presented.

Chapter 2 is a presentation of the underlying physics, mostly focusing on potential flow (PF) theory. Derivations of the governing equations in both Eulerian and Lagrangian perspectives are conducted. The rationale of point, curve, and surface elements is explained, more specifically as it applies to vortex particles, filaments, and quadrilateral sheets. The assembly of elements into lifting line and wake models is detailed. Finally, derivations of induced flow equations as well as derivations of flow field evolution via advection, diffusion, and vortex strength deformation are presented.

Chapter 3 presents the software simulation architecture. The class hierarchy for wake models, wing models, and solution algorithms are presented. This includes fundamental classes such as those for PF elements, wake models, advection equations, diffusion equations, and so on. Computational speed enhancers such as BHTC, FMM, and symmetry models are also detailed.

Chapter 4 is dedicated to performance evaluation and numerical diagnostics. Experimental validation is carried out using the MEXICO HAWT rotor experiments as a basis for comparison. The MEXICO Experiment is a portion of the MexNext Project, a worldwide collaboration whose aim is to assist researchers in validating wind turbine numerical models. In addition, a small validation section is based on the Tjæreborg wind turbine, which made its power curve data public. Numerical diagnostics are performed to ascertain the effects of wake truncation, discretization, and convergence behaviour. Computational acceleration, evolution equations, and wake model selection are covered as well.

Chapter 5 is devoted to applications and initiates investigations of exploring novel wind turbine geometries. The applicability of *LibAero* to various devices is demonstrated. Among these are a standard Danish rotor, a coned HAWT rotor, a swept-bladed HAWT rotor, and a wingleted HAWT rotor.

Chapter 6 presents conclusions about the most pertinent findings, and provides recommendations for future work in line with the priorities and aims of the research path.
Chapter 2

Potential Flow Theory

Chapter 2 explains the foundations of the computational work detailed herein by starting from first principles, namely the Navier-Stokes (NS) equations. A series of derivations are presented that demonstrate the connection between the NS equations and the PF formulation. The Eulerian and Lagrangian perspectives are explained. The Lagrangian perspective gives rise to the advection equation, viscous diffusion, and vortex deformation. The handling of boundary conditions is covered, particularly the lifting line formulation of the wing boundary. Several PF elements and corresponding viscous core models are presented. Finally, a derivation of tree-code algorithms including BHTC and FMM is given.

2.1 Governing Equations

This section presents a step-by-step derivation of the governing equations as they are employed in \textit{LibAero}. It begins with the NS equations in Eulerian form and goes on to provide a detailed listing of the Lagrangian equation models used throughout this work.

2.1.1 Eulerian Perspective

Navier-Stokes Equations

The derivation of the PF methodology employed in this work begins with physical conservation laws. Equation 2.1 is the compressible continuity equation, which governs mass conservation of continuum fluid mechanics. Equation 2.2 is the NS equation
set, which as a three-component vector equation governs conservation of momentum. In the general form given here, $T$ is the stress tensor that typically depends on the local flow velocity, and $f$ is a conservative field force such as gravity [4].

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \quad (2.1)$$

$$\rho \left( \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right) = \mathbf{f} - \nabla p + \nabla \cdot \mathbf{T} \quad (2.2)$$

In Equations 2.1 and 2.2, there are four equations and five unknowns: $\{\mathbf{u}\}_{i=1}^{3}$, $p$, and $\rho$. The missing equation is the energy equation. Given the low rate of thermal energy dissipation due to viscous friction for low-speed aerodynamics problems, the energy equation has insignificant coupling with the momentum equation. For the low-speed aerodynamics that is typical of wind turbines, Mach numbers are less than 10% and incompressibility can be assumed. Thus $\rho$ is assumed constant, and the energy equations can be ignored. Ignoring the energy equation leaves four equations and four unknowns. With constant density, $\rho$ can be taken outside of derivatives for further simplification. As a result, the continuity equation drops the density factor $\rho$ altogether. It becomes the incompressible continuity equation, presented in Equation 2.3. Likewise, the NS equations simplify to the Incompressible Navier-Stokes (INS) equations as shown in Equation 2.4, which in the given form can be considered as Reynolds-averaged equations [36].

$$\nabla \cdot \mathbf{u} = 0 \quad (2.3)$$

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = \frac{\mathbf{f} - \nabla p}{\rho} + (\nu + \nu_T) \nabla^2 \mathbf{u} \quad (2.4)$$

During this step another major simplification was made. The term $\nu_T$, known as turbulent viscosity, was included. The Reynolds stresses that appear in the $T$ term due to Reynolds-averaging are approximated with the turbulent viscosity term, which serves as a zero-equation turbulence model. Turbulent viscosity is generally a variable that depends strongly on the local flow state, and thus requires an additional algebraic equation or PDE to model; this is typical in CFD turbulence modelling. However, this work estimates the turbulent viscosity as a single constant that applies to the regions of the flow where vortex elements are present. As such, the formulation of the unsteady Reynolds-averaged Navier-Stokes (URANS) equations given herein
are governed by a zero-equation turbulence model.

**Scalar and Vector Potentials**

The cornerstone of the potential flow methodology is the introduction of scalar potential \( \phi \) and vector potential \( \psi \) quantities into a restatement of the governing equations. This substitution is based on the Helmholtz decomposition, which states that any sufficiently smooth, rapidly decaying, three-dimensional vector field can be resolved as the superposition of a curl-free (irrotational) component \( \phi \) and a divergence-free component \( \psi \), with each part independently satisfying continuity [37]. The scalar variable \( \phi \) and the three-component vector \( \psi \) replace velocity \( u \) and pressure \( p \) as field variables. Vector potential \( \psi \) is also known as the stream function. For two-dimensional analyses stream function is the preferred terminology since \( \psi \) in two-dimensions is a scalar quantity [4]. Equation 2.5 shows how \( u \) derives from \( \phi \) and \( \psi \). This is called the velocity definition. Calculating \( p \) from \( \phi \) and \( \psi \) becomes a post-processing step, calculated by Bernoulli’s equation in steady potential flow scenarios and calculated by numerical integration otherwise [18].

\[
\mathbf{u} = \nabla \phi + \nabla \times \mathbf{\psi} \tag{2.5}
\]

Taking the divergence of Equation 2.5 reveals the PF representation of the incompressible continuity equation. The influence of the \( \psi \) component vanishes due to the vector calculus identity \( \nabla \cdot \nabla \times \mathbf{A} = 0 \). Finally, the scalar potential \( \phi \) must follow the Laplace equation to represent a physical flow that satisfies the continuity condition.

\[
\nabla \cdot \mathbf{u} = \nabla^2 \phi = 0 \tag{2.6}
\]

Having calculated the divergence of the velocity vector, the next stage is to calculate the curl of the velocity vector \( \mathbf{u} \). Curl of velocity is known as vorticity and denoted as \( \mathbf{\omega} \). Expressing vorticity in potential form, the \( \phi \) component vanishes due to the identity \( \nabla \times \nabla \mathbf{A} = 0 \). The remaining term shows that vorticity equals the negative of the vector Laplacian of \( \psi \).

\[
\mathbf{\omega} \equiv \nabla \times \mathbf{u} = -\nabla^2 \mathbf{\psi} \tag{2.7}
\]

Employing mathematics from potential theory, which was originally developed for electromagnetics, provides additional relationships between flow variables. Equation
2.8 introduces the quantity $\sigma$, known as source density or volumetric source strength. Equation 2.8 shows how the entire flow field is influenced by a region of the space that contains sources or sinks. The gradient of Equation 2.8, given in Equation 2.9, is well-known as Coulomb’s law, which was made famous in the field of electrostatics [38]. It is important to mention that sources and sinks are PF elements that produce mass flow and hence violate the continuity equation at points in space where there is source density. As a result, the usage of sources is restricted to the inside and surface of solid bodies contained in the flow field. The total source density must be zero for conservation of mass to be satisfied.

$$
\phi_x = \frac{1}{4\pi} \int_{\tilde{x} \in V} \frac{\sigma_{\tilde{x}}}{|x - \tilde{x}|} dV \quad (2.8)
$$

$$
\nabla \phi_x = \frac{1}{4\pi} \int_{\tilde{x} \in V} \frac{\sigma_{\tilde{x}}(x - \tilde{x})}{|x - \tilde{x}|^3} dV \quad (2.9)
$$

The vector potential field can be defined in terms of a local quantity as well. That quantity is vorticity $\omega$, as presented in Equation 2.10. Likewise, the gradient form of the equation $\nabla \times \psi$ is well-known as the Biot-Savart law, which is also famous in the field of electromagnetics. The differential form of the Biot-Savart law is given in Equation 2.11 [4].

$$
\psi_x = \frac{1}{4\pi} \int_{\tilde{x} \in V} \frac{\omega_{\tilde{x}}}{|x - \tilde{x}|} dV \quad (2.10)
$$

$$
\nabla \times \psi_x = \frac{1}{4\pi} \int_{\tilde{x} \in V} \frac{\omega_{\tilde{x}} \times (x - \tilde{x})}{|x - \tilde{x}|^3} dV \quad (2.11)
$$

Stemming from the previous derivations, Equations 2.9 and 2.11 are assembled into a general equation for flow field velocity. Flow velocity is defined in terms of the source density field $\sigma_x$ and the vorticity field $\omega_x$. Equation 2.12 demonstrates how the entire flow field is defined in terms of these local field quantities. Equation 2.13 is the discretized form of Equation 2.12 and shows how the entire flow field is defined in terms of discrete PF elements, each with its own influence function $u_{*,x}$ and strength. Source strength of source element $i$ is denoted with $a_i$ and vortex strength of vortex element $j$ is denoted with $\alpha_j$.

$$
u_x = \frac{1}{4\pi} \int_{\tilde{x} \in V} \frac{\sigma_{\tilde{x}}(x - \tilde{x}) + \omega_{\tilde{x}} \times (x - \tilde{x})}{|x - \tilde{x}|^3} dV \quad (2.12)$$
\[
\mathbf{u}_x = \frac{1}{4\pi} \left( \sum_i a_i \mathbf{u}_{i,x} + \sum_j \mathbf{\alpha}_j \times \mathbf{u}_{j,x} \right) \tag{2.13}
\]

**Vorticity Transport**

The curl of the NS equations is known as the vorticity transport equation [4]. Equation 2.14 presents the vorticity transport equation that derives specifically from the zero-order URANS equations as presented in Equation 2.4. The physical interpretation of each term of the vorticity transport equation is labelled in Equation 2.14. The pressure and force terms vanish. First, the vector calculus identity \( \nabla \times \nabla A = 0 \) removes the pressure term since it is a scalar. In addition, \( \nabla \times \mathbf{f} = 0 \) because \( \mathbf{f} \) is a conservative field force such as gravity. As a consequence, to figure the pressure field, pressure must be computed via post-processing of \( \phi \) and \( \psi \), usually through unsteady solution of the unsteady Euler-Bernoulli momentum equation. Equation 2.15 presents the vorticity transport equation in terms of \( \phi \) and \( \psi \). Equation 2.15 is the overall governing equation of the PF methodology detailed herein for the purpose of modelling viscous vortex cores and solving the URANS equations.

\[
\frac{\partial}{\partial t} \nabla^2 \psi + \left( (\nabla \phi + \nabla \times \psi) \cdot \nabla \right) (\nabla^2 \psi) = (\nabla^2 \psi \cdot \nabla) (\nabla \phi) + (\nu + \nu_T) \nabla^4 \psi \tag{2.15}
\]

**Pressure Equation**

Rearranging the zero-order URANS equations and solving for pressure yields Equation 2.16, an intermediate equation. Integrating Equation 2.16 along a curve from any position on the infinite horizon to position \( \mathbf{x} \) results in Equation 2.17, the pressure equation. Unfortunately, post-processing pressures requires numerical integration as presented in 2.17, which is non-trivial. In some cases, pressure can be estimated by Bernoulli’s equation, given in Equation 2.18 [18].

\[
\frac{f - \nabla p}{\rho} = \frac{\partial}{\partial t} (\nabla \phi + \nabla \times \psi) + \frac{1}{2} \nabla |\mathbf{u}|^2 - \mathbf{u} \times \mathbf{\omega} - (\nu + \nu_T) \nabla^2 \mathbf{u} \tag{2.16}
\]
\[
\frac{p_\infty - p_x}{\rho} = \left( \frac{\partial \phi}{\partial t} + \frac{1}{2} |\mathbf{u}|^2 \right) \bigg|_\infty^x + \int_\infty^x \left( \nabla \times \frac{\partial \psi}{\partial t} - \mathbf{u} \times \mathbf{\omega} - (\nu + \nu_T) \nabla^2 \mathbf{u} \right) \cdot d\mathbf{s} \quad (2.17)
\]

\[
\frac{p_\infty - p_x}{\rho} \approx \left( \frac{\partial \phi}{\partial t} + \frac{1}{2} |\mathbf{u}|^2 \right) \bigg|_\infty^x \quad (2.18)
\]

In the pressure equation and Bernoulli’s equation, the time-derivative of scalar potential \( \frac{\partial \phi}{\partial t} \) is a challenge to evaluate. In the case of steady-state flow, the term is zero. Otherwise, the term can be estimated on solid surfaces using Equation 2.19, which is based on a conversion from the Eulerian reference frame to the Lagrangian reference frame of the moving surface [18]. Pressure is typically of most interest on solid surfaces where pressure gives rise to structural forces and motion. In the current work, which is based on lifting line methods, the pressure field is not required for evaluating the flow field and blade forces.

\[
\frac{\partial \phi}{\partial t} = \frac{\partial \phi}{\partial t} \bigg|_{\text{surface}} + \mathbf{u}_{\text{surface}} \cdot \nabla \phi \quad (2.19)
\]

### 2.1.2 Lagrangian Perspective

This section introduces the Lagrangian perspective of the PF methodology that stems from the Eulerian perspective as presented in Section 2.1.1. In the Lagrangian perspective, the governing equations separate into advection, diffusion, and vortex strength deformation equations. These equations and the discretized forms that were employed in this work are detailed in the following subsections. Figure 2.1 depicts rotation, advection, diffusion, and vortex strength deformation as they appear in simulated rotor wakes.

#### Advection

In the Lagrangian perspective of fluid flow, elements carrying fluid properties are advected in the flow because they are force-free. Advection was first introduced as a term of the vorticity transport equation, given in Equation 2.14. The governing equation of motion for any point in a flow is given by Equation 2.20, the advection equation. Specifically, Equation 2.20 presents the differential form of the advection equation.
The discretized first-order [30] and second-order forward Euler advection equations are given in Equation 2.21 and 2.22 respectively. These equations are integral formulations, which are the forms utilized in the numerical models of LibAero. The advection equation applies to all positions on the numerical grid that are free, not bound to a solid body such as a rotor blade. This include vortex particles, vortex filament endpoints, and vortex sheet corner points. In Equation 2.22, the second-order flow acceleration term was computed as a dot product of local flow velocity and local flow velocity gradient.

\[
\frac{\partial x}{\partial t} = u_{xt} \tag{2.20}
\]

\[
x_{t+\Delta t} = x_t + u_{xt} \Delta t + O(\Delta t^2) \tag{2.21}
\]

\[
x_{t+\Delta t} = x_t + u_{xt} \Delta t + \frac{1}{2} (u_{xt} \cdot \nabla u_{xt}) \Delta t^2 + O(\Delta t^3) \tag{2.22}
\]
Diffusion

In any sufficiently detailed flow analysis, transport of properties such as material concentrations, temperature, and vorticity by diffusion become a crucial aspect of the physics model. Diffusion was first introduced as a term of the vorticity transport equation, given in Equation 2.14. The differential form of the diffusion equation is given in Equation 2.23, where \( q \) represents a generic field quantity. The diffusion equation is a second order linear PDE, and its solution depends on initial conditions as well as boundary conditions.

\[
\frac{\partial q}{\partial t} = \lambda \nabla^2 q \tag{2.23}
\]

In the context of fluid flow the diffusivity parameter \( \lambda \) depends to a small extent on the quantity that is diffusing. Momentum diffusivity, as stated in Equation 2.24, stems directly from the viscous term of the NS equations. Mass diffusivity requires an adjustment involving the Schmidt number \( Sc \) and the turbulent Prandtl number \( Pr \), as shown is Equation 2.25. For air, the Schmidt number is approximately 0.75 and the turbulent Prandtl number is approximately 0.85 [39]. In this work, diffusion of vorticity is of interest, which falls in the realm of momentum modelling, thus the simpler equation, Equation 2.24, is employed.

\[
\lambda = \nu + \nu_T \tag{2.24}
\]

\[
\lambda = Sc \cdot \nu + PrT \cdot \nu_T \tag{2.25}
\]

The univariate Gaussian normal distribution is stated in Equation 2.26. The ansatz \( V_t = \sigma_i^2 \) was chosen to satisfy the diffusion equation given in Equation 2.23. The parameter \( V_t \) is the time-dependent variance. The Gaussian distribution is the exact solution of the diffusion equation when the initial distribution is a Dirac delta function \( \delta(\mu) \) at point \( \mu \). Due to central limit theorem, the Gaussian distribution is an approximate solution to the diffusion equation after a sufficient time period, even when the initial distribution is arbitrarily shaped, provided the initial distribution is a compact distribution.

\[
G(x; \mu, V_t) = \frac{1}{\sqrt{2\pi}V_t^{-1/2}} \exp \left( -\frac{1}{2} \frac{(x - \mu)^2}{V_t} \right) \tag{2.26}
\]
Equation 2.27 presents the partial derivative of the time varying Gaussian distribution with respect to time. Equation 2.28 gives the spatial Laplacian of the Gaussian distribution. The two equations differ only by a factor of $\frac{1}{2} \frac{\partial V_i}{\partial t}$. Equation 2.29 demonstrates that the one-dimensional diffusion equation is solved when $V_i = 2\lambda$, shown in Equation 2.28.

$$\frac{\partial G}{\partial t} = G \left[ (x - \mu)^2 V_t^{-2} - V_t^{-1} \right] \frac{1}{2} \frac{\partial V_i}{\partial t} \quad (2.27)$$

$$\frac{\partial^2 G}{\partial x^2} = G \left[ (x - \mu)^2 V_t^{-2} - V_t^{-1} \right] \quad (2.28)$$

$$\frac{\partial G}{\partial t} = \lambda \frac{\partial^2 G}{\partial x^2} \implies \frac{\partial V_i}{\partial t} = 2\lambda \quad (2.29)$$

Directing attention at the three-dimensional diffusion equation as given in Equation 2.23, the multivariate Gaussian normal distribution is summoned. When the multivariate Gaussian is spherical, defined as having equal standard radii in all directions, the multivariate Gaussian can be written as a product of univariate Gaussian distributions, given in Equation 2.30.

$$q_{xyz} = QG(x; \mu_x, V_i)G(y; \mu_y, V_i)G(z; \mu_z, V_i) = QG_x G_y G_z = QG_3 \quad (2.30)$$

Again the time derivative and Laplacian are calculated and shown in Equations 2.31 and 2.32. The decomposition of derivatives reveals that the relationship found in the one-dimensional case holds in the spherical three-dimensional case. Thus variance of the Gaussian distribution evolves linearly at a rate of $2\lambda$.

$$\frac{\partial q}{\partial t} = q \left( \frac{1}{G_x} \frac{\partial G_x}{\partial t} + \frac{1}{G_y} \frac{\partial G_y}{\partial t} + \frac{1}{G_z} \frac{\partial G_z}{\partial t} \right) \quad (2.31)$$

$$\nabla q = q \left( \frac{1}{G_x} \frac{\partial^2 G_x}{\partial x^2} + \frac{1}{G_y} \frac{\partial^2 G_y}{\partial y^2} + \frac{1}{G_z} \frac{\partial^2 G_z}{\partial z^2} \right) \quad (2.32)$$

$$\frac{\partial q}{\partial t} = \lambda \nabla^2 q \implies \frac{\partial V_i}{\partial t} = 2\lambda \quad (2.33)$$

Finally, the local diffusion equation is presented in Equation 2.34. It appears to be a first-order forward Euler diffusion model with a square root to convert between
variance and standard deviation radius. However, this model is the exact solution in fact, since Equation 2.33 prescribes a linear increase in variance.

\[ \sigma_{t+\Delta t} = \sqrt{\sigma_t^2 + 2\lambda \Delta t} \]  

More advanced models such as those proposed by Ramasamy and Leishman take into account the length of a vortex filament [3, 33]. Accounting for filament geometry changes the factor in front of \( \lambda \) from two to four under the assumption that the filament is infinitely long. This work prefers the simple particle core model based on the Dirac delta function, even for filament and sheet elements, since typical numerical model resolutions lead to small filaments and sheets that cannot be assumed to be infinitely long. Future work is recommended to determine whether the relative scale of diffusion lengths and element sizes should influence the diffusion model choice.

**Deformation**

The vorticity transport equation as shown in Equation 2.14 can be rewritten in the Lagrangian perspective using the material derivative operator \( D = \frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla \).

\[ \frac{D\omega}{Dt} = \omega \cdot \nabla \mathbf{u} + (\nu + \nu_T) \nabla^2 \omega \]  

Suppose the distribution of \( \omega \) is of the form \( \alpha_t \kappa_{xt} \) where \( \alpha_t \) is a constant to denote the total integrated quantity of vorticity and \( \kappa_{xt} \) is a time-varying spatial distribution that integrates to unity. Equation 2.36 recasts the vorticity transport equation in terms of \( \alpha_t \) and \( \kappa_{xt} \). Equation 2.36 is then decoupled into a diffusion part on the left and a deformation part on the right, as presented in Equation 2.37.

\[ \alpha_t \frac{D\kappa_{xt}}{Dt} + \kappa_{xt} \frac{d\alpha_t}{dt} = (\alpha_t \cdot \nabla u_{xt}) \kappa_{xt} + (\nu + \nu_T,xt) \alpha_t \nabla^2 \kappa_{xt} \]  

\[ \alpha_t \left[ \frac{D\kappa_{xt}}{Dt} - (\nu + \nu_{T,xt}) \nabla^2 \kappa_{xt} \right] = \kappa_{xt} \left[ -\frac{d\alpha_t}{dt} + (\alpha_t \cdot \nabla u_{xt}) \right] \]  

The left-hand side of Equation 2.37 is the diffusion equation in Lagrangian form. Under the assumptions of the previous section, the left-hand side will equal zero. When the velocity gradient \( \nabla \mathbf{u} \) is zero, the turbulent viscosity \( \nu_T \) is constant, and the viscous core \( \kappa_{xt} \) follows a spherical Gaussian distribution, the left-hand side will cancel. In practice, the velocity gradient is not zero. However, the approximation
is justified when velocity gradient is not wildly variable, for instance in the far wake region. In addition, turbulent viscosity probably varies as well, and the core distribution most likely follows a very unique unknown distribution. With a derivation of the Biot-Savart equation for oblong particle distributions being unknown, the current assumptions have been shown to provide adequate numerical results.

In order to solve the vorticity transport equation, the right-hand side, restated in Equation 2.38 therefore must also equal zero. The discretized vortex particle strength deformation equation of the first-order forward Euler type is given in Equation 2.39. This equation must be explicitly solved for each vortex particle and each iteration. By contrast, vortex filaments and quads depend on equality between circulation values at this time step with those of the previous time step.

\[
\frac{d\alpha_t}{dt} = \alpha_t \cdot \nabla u_{xt} \quad (2.38)
\]

\[
\alpha_{t+\Delta t} = \alpha_t + (\alpha_t \cdot \nabla u_{xt})\Delta t + O(\Delta t^2) \quad (2.39)
\]

### 2.1.3 Boundary Conditions

**Infinite Horizon**

As a result of the methodological assumptions, the infinite horizon experiences the free stream velocity, which may be time-varying. The assumed scenario is an open flow problem where the flow is influenced by lifting surfaces and non-lifting surfaces that may be stationary or in motion, but are finite in position and size. All of the induced flows emanating from these surfaces decrease with distance and have no effect at the infinite boundary. This can be seen as a result of the inverse-square form of PF induction. The inverse-square form of induction causes the infinite boundary conditions to be met automatically, as long as the PF elements of the model are finite in extent. In future research work, if a semi-infinite far-wake model is developed, the finite extent assumption will be violated, and Neumann derivative boundary will be required for as an exit flow condition. The free stream must follow the form

\[
u_{FS} = \nabla \phi_F + \nabla \times \psi_F\]

so that the free stream obeys continuity. The subscript \(F\) denotes the free stream. Equations 2.40 and 2.41 express the boundary conditions at infinity on all horizons.
In CFD, one way to reduce the computational domain and effort is to apply a boundary conditions on the edges of a finite domain. For example, typical upstream and side boundary conditions would be similar to Equations 2.40 and 2.41, but with infinity replace by the domain edge. The downstream edge typically employs a zero-derivative Neumann boundary condition, which assumes the wake is no longer evolving and has become steady. Due to viscous effects, the assumption is not entirely true. The wake continues to evolve until fully developed flow is attained. However, there are computational benefits for the pragmatic numerical assumption that the flow is developing so slowly on the exit boundary and rates of change can be estimated as zero. The computational effort can be devoted to the near wake region for increased precision. Equations 2.42 and 2.43 present the Dirichlet boundary conditions on the domain edge. Equation 2.44 presents the Neumann zero-derivative condition. Work is in progress to develop a far-wake model that emulates the zero-derivative condition for free PF simulation models. Doing so is not as straightforward as in the fixed-domain CFD case.

\[ \phi_{\infty} = \phi_{F,\infty} \]  

(2.40)

\[ \psi_{\infty} = \psi_{F,\infty} \]  

(2.41)

Lifting Lines and Surfaces

Lifting surfaces obey the no-slip condition as shown in Equation 2.45. The flow velocity at a fluid point that meets the solid surface must be the same as the velocity of the moving solid surface, for instance a rotor blade. These boundary assumptions accompany the URANS equations. Here \( \mathbf{u}_S \) is the fluid velocity on surface \( S \) and \( \mathbf{v}_S \) is the solid velocity of the surface.

\[ \phi_{edge} = \phi_{F,edge} \]  

(2.42)

\[ \psi_{edge} = \psi_{F,edge} \]  

(2.43)

\[ \frac{\partial \mathbf{u}_{downstream}}{\partial \mathbf{s}_{streamwise}} = 0 \]  

(2.44)
\[ u_S = v_S \quad (2.45) \]

A less stringent form of the solid surface boundary stems from the continuity equation alone. Equation 2.46 presents the zero-penetration condition, which states that only the surface normal component of flow velocity must equal the solid surface velocity. The surface unit normal vector is written as \( n_S \).

\[ u_S \cdot n_S = v_S \cdot n_S \quad (2.46) \]

This work does not employ the no-slip condition nor the zero-penetration condition directly, but instead makes use of the Prandtl-Weissinger lifting line approximation. Prandtl’s early method employs filaments along the quarter-chord line of the wing and employs zero-penetration control points at the three-quarters chord line. Weissinger later removed the requirement for zero-penetration control points and instead computed angles-of-attack and Reynold’s numbers along the quarter-chord line. Angles-of-attack and Reynold’s number were used to look up airfoil lift and drag values. The latter method is referred in this work as the Prandtl-Weissinger lifting line.

The lifting line is an approximation of the entire lifting surface boundary and requires the lifting surface to be of high aspect ratio. The aspect ratio assumption holds for most aircraft wings and wind turbine blades, and the lifting line approximation has a record of success as demonstrated in Section 1.3, which reviews past literature. This simplification is preferred because the PF method detailed herein is not intended for describing the boundary layer flows that occur on lifting surfaces. To offer an enhanced methodology that models the real physical boundary would compromise the aim of having a speedy aerodynamics tool for MDO purposes. Alternatively, the description of lift and drag forces is achieved through two-dimensional airfoil data with additional corrections for three-dimensional effects. Equation 2.47 presents the Kutta-Joukowski Lift Theorem, which can be considered a lumped representation of the no-slip condition.

\[ \Gamma = \int_A \omega \cdot dA = \frac{1}{2} |\bar{u} - \bar{v}| c C_L = \frac{\nu}{2} R e_{e f f} C_L \quad (2.47) \]

In Equation 2.47 all quantities vary along the lifting line from the root to the tip of the wing or blade. The flow velocity is denoted as \( \bar{u} \) and the velocity of the
solid wing is denoted as \( \bar{v} \). The over-bar indicates that the quantity is averaged over the cross-section of the lifting surface \( A \). The airfoil cross-sectional area is denoted as \( A \) and the \( Re_{eff} \) is the effective Reynolds number. In practice all quantities are computed along the quarter-chord line of the blade.

Non-lifting Surfaces

Non-lifting surfaces obey the no-slip condition as stated in Equation 2.45. However, this methodology does not apply the no-slip condition directly in the non-lifting surface case either, for instance for HAWT towers. Instead collision detection finds PF elements that penetrate solid objects and intelligently relocates them into the flow field. Furthermore, separation effects such as tower shadow are modelled by superimposing a tower shadow element with a prescribed velocity field as carried out by Wang and Coton [29]. Another approach, which has not been implemented in this work, is the approach employed by Willis [18], which employs a panel-type method and specialized matrix solvers based on the fast Fourier transform (FFT). Non-lifting surface effects, however, were not analyzed in the remainder of this work due to the scope and effort constraints of this thesis.

2.2 Potential Flow Elements

PF elements are the basic building blocks of a PF wake and body simulation. PF elements belong to one of two categories, source/sink and vortex. Sources and sinks will be referred to generically as sources since the only mathematical difference between a source and a sink is the sign. The positive sign indicates that the flow originates at the point source and moves away; the negative sign indicates that the flow is drawn toward the point sink and is absorbed. Source elements and vortex elements are models that apply spatial distributions of \( \sigma \) and \( \omega \) respectively to the flow field. As derived in Section 2.1.1, source and vorticity quantities give rise to induced flow via Coulomb’s law and the Biot-Savart Law. The sections below detail the induced flow equations of PF elements and the corresponding viscous core models that were employed in this work.

The source category of PF elements includes the source doublet, usually referred to simply as a doublet. The doublet is the dipole limit of a source and sink of equal and opposite strength as the distance between them approaches zero and the strength
magnitude approaches infinity. A source quadrupole is the next level of the multipole expansion, but is rarely used due to its complexity; its strength is defined by a nine-component tensor. The point vortex itself is defined by a three-component vector. The vortex doublet is unpopular because its strength, like the source quadrupole, is represented by a nine-component tensor. The sign of vortex strength determines the direction of circulation according to the right-hand rule [4].

PF elements are idealized models of induced flow. For example, a point source produces fluid at one singularity point. At all other points in the flow, Equation 2.6 is satisfied, thus providing flow continuity. Likewise a point vortex concentrates rotational effects at one singularity point. At all other points in the flow, Equation 2.7 is satisfied, thus establishing irrotationality. At first glance this seems like an overly strict set of assumptions, but evidence shows that sizable violations of pure PF theory are confined to small boundary layer regions. These regions include the wing surface and the viscous-dominated trailing sheet that emanates from the trailing edge of the wing. The validity of pure PF methods holds for a large family of fluid dynamics problems including wind turbine wake aerodynamics [3].

Violations of the pure PF theory are less of an issue than described due to the recent advances in viscous core and diffusion modelling. These techniques allow the solution of the zero-order URANS equations, which provides higher numerical fidelity than the pure PF methodology, being based solely upon the continuity equation. Although viscous core and diffusion models improve the fidelity of a free vorticity-induced flow field, improvements on top of the pure PF methodology have not offered substantial improvements for modelling surface boundary layers thus far. Recent works indicate that the lifting line approximation is the weak point of PF methods and that the free vortex wake outperforms CFD in many cases [10, 11].

In practice PF elements are not restricted to the point elements that have been discussed so far. To form a PF element into a geometric entity, the point elements are spread over a curve, surface, or volume by integration. The simplest example is that of a point vortex being spread along a straight line segment by integration to create a vortex filament [4]. The inverse perspective is that volumetric vorticity is collected onto a line segment by integration to create the vortex filament. In the particular instance of a vortex filament, the strength vector of the foundational vortex particle must point in the direction along the filament in order to qualify as the classic vortex filament. Derivations of the PF elements used in this work are carried out in the sections below.
Interestingly, some PF geometries show equivalence to one another. A linear doublet filament where the filament direction is the same as the direction of the doublet strength is equivalent to a point source at one endpoint of the doublet filament and a point sink at the other endpoint. A constant strength doublet sheet is equivalent to a vortex closed curve surrounding the edges of the doublet sheet, provided the strengths match. This equivalence is depicted in the lower left-hand portion of Figure 2.2. Further, if the doublet sheet is reduced to a point doublet it becomes clear that a point doublet is equivalent to an infinitesimal vortex ring.

Figure 2.2 presents a visual depiction of a collection of PF elements. The top row of the figure depicts the three most commonly used elements in this work. From left to right, they are the vortex particle, the vortex filament, and the vortex quadrilateral sheet, which is also known as a vortex quad. The bottom-left component of Figure 2.2 presents the equivalence relation between a closed-curve vortex and a doublet sheet. The bottom-right component presents the method that was used to numerically decompose the vortex quad into vortex filaments in the case where there is only trailing circulation and zero shed circulation.
2.2.1 Vortex Particle

The point vortex mentioned previously is also known as a vortex particle. The term vortex particle is preferred in the context of PF elements when a viscous core model is included. Equation 2.48 represents the velocity induced by vortex particle \(i\) at position \(x\) [24]. There is a great deal of similarity between Equation 2.48 and the continuum Biot-Savart law in Equation 2.11. Equation 2.48 concentrates all the vorticity \(\omega\) into the strength \(\alpha\) of the vortex particle. This is akin to integrating the Biot-Savart expression over a Dirac delta distribution located at \(x_A\). The standard substitutions \(r = x - x_A\) and \(R = |r|\) are employed for simplicity. The functions \(U_{VP}\) and \(G_{VP}\) are the induced velocity function and the induced velocity gradient function respectively. The definitions of \(U_{VP}\) and \(G_{VP}\) are used in subsequent derivations.

\[
\mathbf{u} = U_{VP}(x; x_A, \alpha) = \frac{1}{4\pi} \frac{\alpha \times r}{R^3} \tag{2.48}
\]

\[
\nabla \mathbf{u} = G_{VP}(x; x_A, \alpha) = \frac{1}{4\pi} \frac{\alpha \times (R^2I - 3rr^T)}{R^5} \tag{2.49}
\]

2.2.2 Vortex Filament

Condensing vorticity onto a line segment where the vorticity direction shares direction with the line segment creates a vortex filament. This arrangement is derived by integrating a vortex particle along the line segment from Point \(A\) to Point \(B\) [4]. Specifically, the vortex filament covered here has constant strength. In the case of a vortex filament, strength is known as circulation. Figure 2.3 shows how vector expressions are transformed into trigonometric expressions on a two-dimensional plane. It is noteworthy that while the vortex filament is a simple vortex element, its Biot-Savart equation is fairly complex. It becomes even more challenging to derive closed-form equations to represent PF elements with complex geometries such as vortex sheets and rings.

Equation 2.50 is the starting equation for deriving the Biot-Savart kernel of a vortex filament. The derivation begins by integrating to calculate the limit value produced by a line segment with infinitely many vortex particles of infinitesimal strength. Equations 2.51, 2.52, and 2.53 offer trigonometric transformations that facilitate analytical integration. Equations 2.54, 2.55, and 2.56 show further integration and simplification steps, resulting in the induced velocity equation of a vortex filament.
Figure 2.3: Vortex filament

The introduced variables $r$ and $\alpha$ can be understood as being either $r_A$ and $\alpha_A$ or $r_B$ and $\alpha_B$ as shown in Figure 2.3; the numerical outcome is independent of the choice.

$$u = \frac{1}{4\pi} \int_{\tilde{x} = x_A}^{\tilde{x} = x_B} \frac{\Gamma d\tilde{x} \times (x - \tilde{x})}{|x - \tilde{x}|^3} d\tilde{x} \quad (2.50)$$

$$a = r \sin \alpha \quad (2.51)$$

$$b = r \cos \alpha = a \cot \alpha \quad (2.52)$$

$$d\tilde{x} = db = -\frac{a}{\sin^2 \alpha} d\alpha = -\frac{r}{\sin \alpha} \quad (2.53)$$

$$u = \frac{\Gamma_i}{4\pi} \int_{\tilde{x} = x_A}^{\tilde{x} = x_B} \frac{\sin \alpha d\alpha}{|a|} \cdot \frac{r_A \times r_B}{|r_A \times r_B|} \quad (2.54)$$

$$u = \frac{\Gamma_i}{4\pi} (\cos \alpha_A - \cos \alpha_B) \frac{r_A \times r_B}{|r_A \times r_B|} \quad (2.55)$$

$$u = \frac{\Gamma}{4\pi} \left[ (r_B - r_A) \cdot \left( \frac{r_A}{|r_A|} - \frac{r_B}{|r_B|} \right) \right] \frac{r_A \times r_B}{|r_A \times r_B|^2} \quad (2.56)$$

Using the Gram determinant identity, $|a \times b|^2 = |a|^2 |b|^2 - (a \cdot b)^2$ where $a$ and
\( \mathbf{b} \) are arbitrary vectors, allows the Biot-Savart equation of the vortex filament to be cast in the relatively simple form, given in Equation 2.57.

\[
\mathbf{u} = U_{VF}(x; x_A, x_B, \Gamma) = \frac{\Gamma}{4\pi} \left( \frac{1}{R_A} + \frac{1}{R_B} \right) \frac{\mathbf{r}_A \times \mathbf{r}_B}{R_A R_B + \mathbf{r}_A \cdot \mathbf{r}_B} \tag{2.57}
\]

Due to the algebraic complexity exhibited here, the closed-form expression of velocity gradient has been omitted. Induced velocity gradients are computed using the software SymPy, an open-source computer algebra library for the Python scripting language. The resulting expressions are converted into C++ code.

### 2.2.3 Vortex Quad

With Biot-Savart derivations of the vortex particle and the constant circulation vortex filament, the next PF element in sequence is a vortex sheet. Specifically, the vortex sheet employed in this work is a quadrilateral sheet element without restriction on the positions of the corner points. This means the vortex sheet described herein is not planar but rather a bilinear surface known as a vortex quad. The rationale of free corner points is to avoid unnecessary geometric restrictions that would complicate the three-dimensional dynamics. Basom and Maughmer discuss some of the numerical leakage problems that occur when approximating quad elements with planar sheet elements and suggest that increasing the number of planar elements is the appropriate solution [40]. Thus, the disadvantage of the quad element is that an analytical expression of the Biot-Savart law is usually unattainable, which requires dividing the quad into sub-elements for numerical integration. Likewise, the disadvantage of the planar element approximation is an increase in the number of computational elements. One possibility for future work is to conduct a comparison of the two approaches.

The vortex quad has circulation spread with linear variation from edge \( AB \) to edge \( DC \), and another circulation value spread with linear variation between \( AD \) and \( BC \). A depiction is given in Figure 2.4. A derivation of the induced velocity comes by integrating the Biot-Savart equation of a vortex filament over the sheet surface in the \( AB \)-to-\( DC \) span, doing the same in the \( AD \)-to-\( BC \) span, and summing these two components. In practice the integral is computed numerically by generating \( M \) temporary vortex filaments along the \( AB \)-to-\( DC \) span and \( N \) filaments along the \( AB \)-to-\( DC \) span. The numbers \( M \) and \( N \) can be preset or adaptively adjusted to provide a smooth element depending on the viscous core radius. In Figure 2.4, \( M \) equals
five and $N$ equals three. The circulations of the temporary filaments are depicted to demonstrate how linear variation of circulations from edge to opposing edge is achieved. In the $AB$-to-$DC$ span, the circulation is depicted with a change in sign.

Figure 2.5 demonstrates that the overlapping diffuse core distributions of vortex filaments can be superimposed to form a smooth or nearly smooth core distribution for the resulting vortex quad. The top section of the figure presents a low resolution model based on spacing viscous cores apart by two standard deviations. The bottom section of the figure presents a medium resolution model based on spacing cores at one standard deviation apart.

### 2.2.4 Source Particle

Having discussed vortex elements and their induced flow equations, source and doublet elements deserve mention, even though they are not employed in this work at the current stage. For vortex elements, the induced flow equations derive from the Biot-Savart Law. For source elements, the induced flow equations follow Coulomb’s Law. Equation 2.58 is Coulomb’s Law of a point source [4]. Apart from Equation 2.58 being a scalar equation without any cross-product term, there is a noticeable similarity to the Biot-Savart Law of the point vortex as given in Equation 2.48.
Doublet elements are another subcategory of PF elements that are commonly used in PF simulations. The doublet particle described here belongs to the source family of PF elements. There are doublet vortexes as well, but that line is not pursued in this work. The doublet is a limit of a source and sink that approach one another closer and closer while increasing in strength. The source and sink may be points, lines, or sheets, yielding a doublet particle, filament, or sheet respectively. The induced flow equation of the point doublet is given in Equation 2.59 [4]. It is expressed in integral form using scalar potential $\phi$ for the sake of simplicity. The electromagnetic analog is the electric dipole equation.

$$\phi = \Phi_{DP}(x; x_A, b) = -\frac{1}{4\pi} \frac{b \cdot r}{R^3}$$  \hspace{1cm} (2.59)
two layers of doublet sheets in the near wake attached to the trailing edge of wings [18]. Katz and Plotkin also employ double sheet wake models in their popular text [4]. However, *LibAero* currently focuses on vortex based methods and attempts to gain high geometric resolution with vortex quads, and hence does not employ doublet elements at the current stage of development.

### 2.2.6 Free Stream

The superposition principle that underlies the PF methodology applies to free stream elements as well. One additional constraint is that the PF elements in the flow field cannot couple with the parameters that define the free stream. This means the free stream is not independent of the net flow field. In PF methods, superposition of PF element-induced flows is a standard assumption and that applies to the free stream as well [4]. In the case of PF methods with viscous core modelling, numerous authors apply the same assumption [30, 24, 25, 28]. Free stream models can be steady, unsteady, or periodic. The simplest free stream is the steady free stream and it was employed universally in this work. The steady free stream provides velocity and velocity gradient according to Equations 2.60 and 2.61.

\[
\mathbf{u} = U_{SF} = (u_{SF}, v_{SF}, w_{SF})
\]

\[
\nabla \mathbf{u} = G_{SF} = 0_{3x3}
\]

### 2.3 Vortex Core Models

In the preceding sections non-diffuse PF elements were described and their Biot-Savart equations were derived. This section introduces the implementation of diffuse viscous core distributions. Since real fluid flows undergo diffusion processes, vorticity spreads. Introducing a spatial distribution of vorticity improves the physical model fidelity. Furthermore, viscous core modelling removes singularity points from the Biot-Savart equations, which improves robustness of the numerical model.

On the flip side, the PF approach is strictly valid only in irrotational regions of the flow field since the viscous models are approximations. With diffuse core models, and especially when the core distribution has infinite tails, much of the flow field has vorticity. In regions with vorticity, the governing equations are not the pure PF
equations, but rather are the URANS equations. As such, numerical accuracy must always be under consideration when implementing a low-order URANS model such as the zero-equation model employed in this work.

An illustration of the relationship between vortex core distributions and the vorticity transport equation is to consider a flow field described by several diffuse vortex elements. First, one chooses several collocation points in the flow to compute flow velocity, velocity gradient, and vorticity via the Biot-Savart law. By this method of computing, the vorticity value depends largely on far-away flow variables that induce the flow field. Next, one can compute vorticity at the same collocation points from the spatial distributions of each vortex element’s strength. By this method of calculation, the vorticity value depends almost entirely on nearby vortex elements since typical core distributions have narrow tails. The exact solution to the governing equations requires the vorticity values produced by each approach to be equal. In practice, it is not usually possible to make this comparison, since many of the core models are defined only by their Biot-Savart equations, and thus are challenging to back-calculate.

The following derivation demonstrates how a distributed core model can give rise to a Biot-Savart equation that is a slight modification of the original non-diffuse PF element Biot-Savart equation. Equation 2.62 derives from the volumetric Biot-Savart law in Equation 2.11, but replaces vorticity $\omega$ with vortex strength multiplied by a spatial distribution function $\alpha \kappa_\tilde{x}$.

$$\mathbf{u} = \frac{1}{4\pi} \int_{\tilde{x} \in V} \frac{\alpha \times (\mathbf{x} - \tilde{x})}{|\mathbf{x} - \tilde{x}|^3} \kappa_\tilde{x} dV \quad (2.62)$$

For certain PF element and core model combinations, Equation 2.62 can be reduced to Equation 2.63. In the equation, $U_{0,x}$ is the Biot-Savart influence of the core-less PF element and $K_x$ is a spatial function that multiplies by the Biot-Savart equation of the core-less element.

$$\mathbf{u} = \begin{cases} 0 & \text{for } U_{0,x} \text{ singularity points} \\ U_{0,x}K_x & \text{elsewhere} \end{cases} \quad (2.63)$$

When the Biot-Savart equation of a diffuse PF element follows the form in Equation 2.63, the velocity gradient is computed by chain rule as shown in Equation 2.64.
\[ \nabla u = \begin{cases} 
0 & \text{for } U_{0,x} \text{ singularity points} \\
\nabla U_{0,x} K_x + U_{0,x} \nabla K_x & \text{elsewhere} 
\end{cases} \tag{2.64} \]

The following sections provide the details of several vortex core models. Of these, the Gaussian and Vatistas cores are core models that follow the decoupled form of Equations 2.63 and 2.64.

### 2.3.1 Gaussian Vortex Particle Core

The most widely accepted PF element viscous core distribution is the Gaussian distribution. Although a wide range of spatial distributions are feasible as core models, and physical vortex core distributions are still the subject of active research [3], the Gaussian distribution is the most compatible with simple diffusion processes. When a Gaussian distribution undergoes diffusion, it remains a Gaussian distribution, but with increased variance parameter. The Gaussian distribution is not necessarily the true vorticity distribution based on flow physics, however. Inertial terms and turbulence effects can give rise to exotic vortex core distributions, in the case of a trailing wing tip filament for instance [3]. The Gaussian distribution of a vortex particle is given in Equation 2.65, and is known as the Gaussian kernel. The kernel equation corresponds to the situation when the total quantity that is diffusing is unity, \( Q = 1 \).

\[
\kappa = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{R^2}{2\sigma^2}} \tag{2.65}
\]

Substitution into Equation 2.62 yields the Biot-Savart equation of the Gaussian-distributed vortex particle, given in Equation 2.66 [11]. The equation follows the decomposed form given in Equation 2.63, with the value of \( K \) being equal to \( 1 - \exp\left\{-\frac{R^3}{\sigma^3}\right\} \). The \( R^3 \) term was already computed in the denominator of the left-hand portion, thus the term can be reused and for a subtle computational benefit.

\[
u = U_{VP,Gaussian}(x; x_A, \alpha, \sigma) = \frac{1}{4\pi} \frac{\alpha \times r}{R^3} \left(1 - \exp\left\{-\frac{R^3}{\sigma^3}\right\}\right) \tag{2.66}
\]
2.3.2 Garrel Vortex Filament Core

The vortex filament core model obtained from van Garrel [30] is referred to as the Garrel core model in this work. Its Biot-Savart equation is given below in Equation 2.67. Notice the denominator has an extra term that is guaranteed to be positive as long as the vortex filament has non-zero length. This removes the singularity from the expression. The symbol $\tilde{\sigma}$ is used in place of $\sigma$ since the Garrel core radius is not equivalent to Gaussian standard deviation. It may, in some cases, be possible to find a proportionality factor that relates the Garrel smoothing distance to the Gaussian smoothing distance, although the distributions as a whole are different. This avenue was not pursued thus far, however. The reason to part from Gaussian core models is the difficulty in deriving a Biot-Savart equation for the Gaussian core of particular element geometries. On the other hand, it is easy to make a modification to the Biot-Savart equation, such as inserting an additional positive term into the denominator. The Garrel core model is an example of this kind of pragmatic adjustment, although it does not follow the decomposed form given in Equation 2.63.

$$u = U_{VF,Garrel}(\mathbf{x}; \mathbf{x}_A, \mathbf{x}_B, \Gamma,\tilde{\sigma}) = \frac{\Gamma}{4\pi} \frac{(R_A + R_B)(r_A \times r_B)}{R_A R_B (R_A r_B + r_A \cdot r_B) + \tilde{\sigma}^2 R_{AB}^2}$$ (2.67)

2.3.3 Vatistas Family of Core Models

The Vatistas family of core models has a parameter $n$ that allows different distributions to be achieved from the same generic function. When $n = 1$ the Vatistas core is known as the Scully core model [28]. As $n$ approaches infinity the Vatistas core becomes the Rankine core model [3]. The setting $n = 2$ is popular because it closely approximates the Gaussian core model [3]. The Vatistas core function is given is Equation 2.68 [41]. The parameter $\eta$ is the perpendicular distance from the line of the element to the point where induced flow is computed.

$$K(\eta; n) = \frac{\eta^2}{(\eta^{2n} + \tilde{\sigma}^{2n})^{1/n}}$$ (2.68)

The Vatistas family of cores is applicable only to filament- and particle-type elements. For vortex filaments, the line of the element is in the direction of the filament. For vortex particles, the line of the element points in the direction of the strength vector. The definitions of $\eta$ are given in Equation 2.69 and 2.70 for filaments and
particles respectively.

\[ \eta = |\mathbf{x} - \mathbf{x}_A - \text{proj}(\mathbf{x} - \mathbf{x}_A, \Gamma(\mathbf{x} - \mathbf{x}_A))| \quad (2.69) \]

\[ \eta = |\mathbf{x} - \mathbf{x}_A - \text{proj}(\mathbf{x}, \Gamma(\mathbf{x}_B - \mathbf{x}_A))| \quad (2.70) \]

### 2.4 Prandtl-Weissinger Lifting Line

Continuing from Equation 2.47, the discretized form of the lifting line equations are given in Equations 2.71 and 2.72. Both equations are statements of circulation conservation. Equations 2.71 and 2.72, as well as wake models, are constructed to ensure that the three Helmholtz criteria are satisfied:

1. A vortex filament must have constant circulation strength. In order to compose advanced wake models such as the horseshoe model, filaments are superimposed and the total circulation value is employed in computations for the sake of efficiency.

2. A vortex filament cannot start or end in the finite region of the fluid. Either vortex filaments must form closed path or extend to infinity. In this work, the wake models extend to infinity in theory, while in practice the wakes are truncated downstream.

3. Each vortex filament path maintains its circulation value over time as the path advects and diffuses in the evolving flow field.

Equation 2.71 conserves circulation about the flow field at any given instant and is used for computing circulation values trailing at the blade-wake interface; this derives from Helmholtz’s second criterion. Equation 2.72 conserves circulation across time steps as laid out in the third criterion of Helmholtz; it is used to compute circulation values in the shed direction of the blade-wake interface. The subscript \( LL \) signifies the lifting line. In the equations, \( N \) is the number of vortex filaments that comprise the lifting line. The ordinal \( i \) is used to denote the particular element, with \( i = 0 \) representing the lifting line or trailing element nearest to the root [4]. The equations that govern circulation values along the blade based on airfoil coefficients and interpolation are detailed in Section 3.5.1.
\[ \Gamma_{Travel,i} = \begin{cases} \Gamma_{LL,i} & \text{for } i = 0 \\ \Gamma_{LL,i} - \Gamma_{LL,i-1} & \text{for } 0 < i < N \\ -\Gamma_{LL,i-1} & \text{for } i = N \end{cases} \tag{2.71} \]

\[ \Gamma_{Shed,i} = \Gamma_{LL,i} - \Gamma_{LL,i,Previous} \tag{2.72} \]

## 2.5 Tree-Code Algorithms

The induced flow computations of the N-body dynamics can be greatly accelerated using numerical approximation techniques. In the standard N-body problem, there are N objects, which may be stars, galaxies, charged particles, or PF elements. In the first three examples, force is the induced quantity, whereas flow velocity is the induced quantity in the case of PF elements. In general, the induced quantity is called influence. Each object induces its influence on all others and in turn is influenced by all other objects. In order to compute the influence on one object, the direct algorithm evaluates the force contribution from each other object, and uses superposition to compute the total induced influence. This induced influence step requires a number of operations of order \( O(N) \). One-by-one, each object has its total induced influence computed, and the object is evolved according to the motion equation, which involves position, velocity, and acceleration. The evolution step is \( O(N) \) as well. One \( O(N) \) step nested within another \( O(N) \) step makes an overall computational complexity of \( O(N^2) \).

Barnes-Hut tree-code (BHTC) and fast multipole method (FMM) are two related methods that reduce the computational complexity to \( O(N \log N) \) and nearly \( O(N) \) respectively. Both methods are based on the tree-code algorithm, which sorts the objects into a spatial tree. Both tree-code methods rely on having a fast approximation of the induction equation for objects whose tree node location is far away. Multipole expansions serve this purpose, and in this work only the monopole was employed. For nearby objects, the exact induction equation is employed [22].

Figure 2.6 depicts the binary tree data structure used in LibAero, with the exception that the figure is two-dimensional and LibAero is a three-dimensional code. The dots represent vortex particles, the line segments are vortex filaments, and the quadrilaterals are vortex quads. The diagram presents randomized elements for ease.
of explanation, whereas actual wake structure is ordered.

In the figure, the maximum number of elements in the leaf region is three. The computational part of this work uses a setting of four elements per leaf region. The thick dashed line depicts the topmost binary division. As the dashed line becomes thinner, it represents subdivisions. The smallest subdivision is known as a leaf node or leaf region. The location and orientation of the dividing line is determined from sample statistics on the elements whose centres are contained within the region that is being divided. The equations that govern the operation of the tree-code are detailed in the following paragraphs. This strategy of using a binary tree with sampling is an attempt to balance the tree and much as possible for theoretical performance improvements [22]. The cross symbol marks the centre of non-particle elements since the centre is essential for sorting the elements into their appropriate bins.

![Figure 2.6: N-body binary tree](image)

In this work a binary tree data structure was employed instead of the more usual octree data structure that is used in three-dimensional N-body simulations. The rationale of this choice is the oblong shape of the rotor wake, which does not lead to balancing and good performance in cubical octree-based algorithms. The following equations outline the steps of building the binary tree and carrying out BHTC or FMM computations. Equations 2.73, 2.74, and 2.75 give methods of computing the centre $c$, element bounding sphere diameter $d$, and monopole strength $s$ respectively.
for element $i$, provided that element $i$ is a vortex particle, filament, or quad. Since the PF element core is diffuse with infinite tails, to make the bounding sphere finite, a practical truncation radius must be selected. The $A$ parameter defines a cutoff factor and is usually set to a value of two standard deviations.

$$c_i = \begin{cases} x_{Ai} & \text{for particles} \\ \frac{1}{2}(x_{Ai} + x_{Bi}) & \text{for filaments} \\ \frac{1}{2}(x_{Ai} + x_{Bi} + x_{Ci} + x_{Di}) & \text{for quads} \end{cases}$$

$$d_i = \begin{cases} 2A\sigma_i & \text{for particles} \\ |x_{Bi} - x_{Ai}| + 2A\sigma_i & \text{for filaments} \\ \max\left(|x_{Ci} - x_{Ai}|, |x_{Di} - x_{Bi}|\right) + 2A\sigma_i & \text{for quads} \end{cases}$$

$$s_i = \begin{cases} \alpha_i & \text{for particles} \\ \Gamma_i(x_{Bi} - x_{Ai}) & \text{for filaments} \\ \Gamma_{ABi}(x_{Bi} - x_{Ai}) + \Gamma_{DCi}(x_{Ci} - x_{Di})/2 + \Gamma_{ADi}(x_{Di} - x_{Ai}) + \Gamma_{BCi}(x_{Ci} - x_{Bi}) & \text{for quads} \end{cases}$$

As was the case for elements, each node of the binary tree has a centre, a bounding sphere radius, and a multipole strength. If the node has more than the fixed limit for the number of elements in one node, usually set to a value of four, the elements are pushed down to sub-nodes labelled left and right. The labels left and right are completely arbitrary. Equations 2.76, 2.77, and 2.78 give methods of computing the centre $C$, bounding sphere diameter $D$, and monopole strength $S$ respectively of a particular node. The centres and diameters are sample-estimated from contained items when descending the tree on the initial build routine. Specifically, the nodal diameter is the smallest diameter of a bounding sphere that would hold all the elements in the node. The practice of using bounding spheres overcomes the reliance of the tree-code algorithm on point elements and allows non-point elements to be used conservatively in the approximation formulas. Whereas the centre and diameter are computed while descending the tree in the initial build routine, the monopole strength is computed while ascending the tree in the step that immediately follows. While ascending, Equation 2.79 is used to compute a more precise estimate of the node centre. The quantities $S_{left}$ and $S_{right}$ are zero if the sub-node does not exist,
in other words being empty and having no monopole strength.

\[ C_{\text{node}} \approx \sum_{i \in \text{Node}} c_i \]  

(2.76)

\[ D_{\text{node}} = \max_{i,j \in \text{Node}} \left( |c_i - c_j| + \frac{A}{2}(d_i + d_j) \right) \]  

(2.77)

\[ S_{\text{node}} = S_{\text{left}} + S_{\text{right}} + \sum_{i \in \text{Node}} s_i \]  

(2.78)

\[ C_{\text{node}} = \frac{1}{|S_{\text{node}}|} \left( |S_{\text{left}}| C_{\text{left}} + |S_{\text{right}}| C_{\text{right}} + \sum_{i \in \text{Node}} |s_i| c_i \right) \]  

(2.79)

The BHTC method of computing induced flow velocity is given in Equation 2.80. Like the descending and ascending build routines, the computing induced flow is recursive; that means \( u_{\text{left}} \) and \( u_{\text{right}} \) are the same function call as \( u_{\text{Node}} \), but acting on the sub-node instead. Finally, Equation 2.81 unites the free stream with the BHTC approximation of PF element-induced flow to compute the flow at any Point \( x \) in the flow field. The parameter \( B \) is a cutoff ratio that determines when to employ the approximation formula based on nodal summary values, rather than individual element information. The tree-code cutoff is a ratio of the distance between the point in question and the centre of the node or element acting on that point to the diameter of the bounding sphere that would surround the node or element. In other words, looking outward from Point \( x \), if the node or element occupies a small solid angle, the approximation formula can be employed. Typical \( B \) values range from 0.25 to 2.0.

\[ u_{BHTC,\text{Node}} = \begin{cases} \frac{1}{4\pi} \frac{S_{\text{node}} \times (x - C_{\text{Node}})}{|x - C_{\text{Node}}|^2} & \text{if } \frac{|x - C_{\text{Node}}|}{D_{\text{node}}} > B \\ u_{BHTC,\text{left}} + u_{BHTC,\text{right}} + \sum_{i \in \text{Node}} u_i & \text{otherwise} \end{cases} \]  

(2.80)

\[ u = U_F + u_{BHTC,\text{Trunk}} \]  

(2.81)

The FMM method of computing induced flow velocity involves first precomputing the far-away lumped influence \( u \) at the centre of each populated node \( C_{\text{Node}} \) using Equation 2.82. Equation 2.82 is similar to Equation 2.80, but lacks the summation term for computing the influence of nearby elements. As the far-away lumped influence is computed for a node, a list of excluded elements is kept under the name
Neighbours and is stored within each node. Since the elements that belong to a particular node are close together, they share the same far-away lumped influence approximately. Finally, the excluded elements are used directly to compute the nearby influence, which is added to the far-away lumped influence and the free stream to produce the flow velocity at the desired point as detailed in Equation 2.83.

\[
\begin{align*}
    u_{FMM,Node} &= \begin{cases} 
        \frac{1}{4\pi} \frac{S_{Node} \times (u - C_{Node})}{|u - C_{Node}|^3} & \text{if } \frac{|x - C_{Node}|}{D_{node}} > B \\
        u_{FMM,left} + u_{FMM,right} & \text{otherwise}
    \end{cases} 
\end{align*}
\]

(2.82)

\[
    u = U_F + u_{FMM,Trunk} + \sum_{i \in \text{Neighbours}} u_i
\]

(2.83)

2.6 Relaxation Iteration

The relaxation method is a technique for accelerating the convergence of an iterative problem. The method was advanced by Frankel and Young independently in 1950. When iteratively solving an equation set on a numerical grid, sometimes the current values approach the converged values systematically from one side by inching closer and closer; other times the values oscillate and gradually approach a converged value that lies in the middle; and in some instances the behaviour is more advanced. For problems where convergence involves a one-sided approach, the over-relaxation method causes values to overshoot the predicted value. The corrected values propagate throughout the numerical grid and accelerate convergence if the relaxation parameter is chosen suitably. For over-relaxation, the relaxation parameter is greater than unity. For problems where convergence involves an oscillatory approach, under-relaxation, which is also known as smoothing, dampens the oscillation and accelerates convergence. Here, the relaxation parameter is less than unity. Equation 2.84 presents the relaxation equation for correcting predicted values using the generic numerical value \( Q \). The factor \( \alpha \) is the relaxation parameter [36].

\[
    Q_{i+1,Corrected} = \alpha Q_{i,Corrected} + (1 - \alpha) Q_{i+1,\text{Predicted}}
\]

(2.84)
Chapter 3

Simulation Architecture

Chapter 3 is an overview of the simulation architecture employed in *LibAero*. This chapter explains how the simulation was organized: which aspects utilize OOP and which aspects utilize a procedural programming approach. Such design choices were the subject of many individual tradeoffs while the central goal was to produce an extensible, maintainable, and modifiable code base. In this way, new simulation runs can be developed with the maximum reuse of code.

The abstract breakdown of simulation classes calls for four categories. These categories are aerodynamic data, aerodynamic algorithms, wing data, and wing algorithms. The term *wing* is used as a generalized term that includes wind turbine rotor blades. In addition, there are miscellaneous functions for spatial transformations, data file input, and data file output, as well as one unique data class, *Influence*, which belongs neither to the aerodynamics nor the wing category. Aerodynamic data and algorithms pertain to lifting line and wake models. Wing data and algorithms apply to the motion state of the wing as well as solid-fluid interface properties such as angle-of-attack and Reynolds number. Solid mechanics algorithms such as aeroelasticity models can be incorporated as wing algorithms, although this work focuses on rigid-body rotation of a rotor.

The OOP paradigm allows algorithms to be bundled with data as class methods. The design choice of keeping data and algorithms separate was made after experimenting with *integrated* and *separated* options and finding better modularity and expressibility in the *separated* design. Literature concerning the object-oriented design of numerical codes is relatively weak outside of FEM codes where the crucial feature is the sharing of element end-nodes. While *LibAero* utilizes end-node sharing, the defining feature of *LibAero* is that all PF elements influence the behaviour
of all others through flow induction. In FEM, only conjoining neighbours influence one another, and this allows FEM problems to be formulated and solved as a linear system in many cases.

## 3.1 Influence

![Influence class hierarchy](image)

The *Influence* class describes the flow conditions at one point in space. *Influence* in its simplest form stores only the local flow velocity, called the *VelocityOnly* model. A more advanced *Influence* stores the local flow velocity as well as the gradient of the flow velocity, under the name *VelocityGradient*. The extra local information gives rise to additional functionality. Vorticity can be computed using Equation 2.7. Furthermore, flow acceleration can be computed using Equation 3.1.

\[
a = u \cdot \nabla u
\]  

(3.1)

Numerous models used in PF calculations depend on velocity gradient. The second order advection model given in Equation 3.5 requires the flow acceleration, which is computed as the dot product of flow velocity with flow velocity gradient. The first-order strength deformation from Equation 2.39 used in conjunction with vortex particles requires velocity gradient as well. At the same time, it is possible with *LibAero* to construct lifting line and wake models that do not require velocity gradient. The benefit of avoiding velocity gradient is the computational savings of not evaluating nine-term velocity gradients via the Biot-Savart law. The benefit of including velocity gradient calculations is the higher order and improved fidelity of advection and deformation models.
3.2 Aerodynamic Data Classes

This section describes classes that pertain to PF elements, hence the lifting line and the free wake. An abstract base class for such objects is named AeroObject. An AeroObject is able to evolve according to the flow field. The classes described in the following sections are derived from the AeroObject parent class.

3.2.1 AeroElement

The AeroElement class is the base class for PF elements. Typical child classes that were used throughout this work are VortexParticle, VortexFilament, and VortexQuad. The free stream does not inherit from AeroElement; further information regarding the free stream is given in Section 3.3.1. A proposed future addition to LibAero is a BEM wake model, which should be a categorized as an AeroElement.

3.2.2 AeroNode

The AeroNode class is the basic building block of AeroElement. AeroElement contains an array of AeroNodes. Typical implementations of AeroNode are Point, Core, and Strength. Point contains the position three-component vector. Core contains the viscous core radius of the diffuse PF element. Strength contains the PF element induction strength. The Strength of a vortex particle is a three-component vector whereas Strength of vortex filaments is a scalar.
The reason for decomposing AeroElement into an array of AeroNodes is to allow advection models, diffusion models, and strength deformation models to be used interchangeably on different AeroElement types. For instance, advection of Point is the same regardless of its belonging to a vortex particle, vortex filament, or vortex quad. Integer tags are contained in each AeroNode to serve as pointers to shared AeroNodes, as well as previous and subsequent states of the AeroNode. This approach allows all pertinent information to be accessed by the equation models. Integer tags were chosen over pointers to facilitate deep-copying of class objects; the advantages and disadvantages of this approach are covered in Section 3.6.

3.2.3 AeroAssembly

The AeroAssembly class is a base class for wake models. AeroAssembly is built as an array of AeroElements and contains all of the elements in the flow at a point in time. Functions are provided to help set the integer tags shared, previous, and subsequent AeroNodes. If an AeroNode shares with another, the advection model advects only the master AeroNode and sets the slave AeroNode position equal to the position of the master AeroNode. This is further explained in Section 3.3, which covers aerodynamic algorithm classes. VortexFilament and VortexQuad are two elements that rely on shared nodes. VortexFilaments appear in chains while VortexQuads appear in meshes; neighbours share endpoints and corner points respectively.

Derived classes of AeroAssembly have been prepared with the sharing structure
pre-defined for ease-of-use. Some derived classes contain only trailing vortex elements as needed in steady-state simulations. Other derived classes contain both trailing and shed vortex elements, which can be used in steady and unsteady simulations. PrandtlSteadyModel is a pure VortexFilament wake model as described by Prandtl and developed further by Weissinger since it contains only trailing vortex filaments [4, 19]. PrandtlSteadyParticleModel is a modification to the PrandtlSteadyModel that requires VortexFilaments to transition into VortexParticles in the far wake; the transition layer is pre-set by the user. QuadSteadyModel is similar to PrandtlSteadyModel, but is based on the VortexQuad element instead of VortexFilament. QuadSteadyParticleModel extends the QuadSteadyModel with VortexParticles in the far wake region. These wake models have prescribed connectivity and shape, while size parameters such as the number of layers of filaments and the number of layers of particles are defined by the user at simulation time.

3.2.4 AeroState

The AeroState class is a collection of all aerodynamic numerical data. AeroState contains an array of AeroAssembly objects, each one representing a snapshot in time. In unsteady and periodic simulation cases, the difference from steady-state is the connectivity of previous and following states contained in AeroNodes that belong to different snapshots. Because this work focuses on steady-state simulations and validations, the array of snapshots required only one entry for the current snapshot. The functionality for multigrid computational acceleration through additional linked
snapshots is also present in AeroState and has been tried successfully. However, multigrid results are not presented due to scope limitations.

### 3.3 Aerodynamic Algorithm Classes

#### 3.3.1 FreeStream

The FreeStream class is an abstract base class for computing Influence at position $\mathbf{x}$ in the flow. Its purpose is reserved for the representation of the free stream flow before any PF elements are superimposed. Currently there is one child class, SteadyFreeStream. SteadyFreeStream returns the same velocity regardless of $\mathbf{x}$. If summoned, the velocity gradient returned by a SteadyFreeStream object is zero. In this way, the SteadyFreeStream class is compatible with both Influence subtypes, VelocityOnly and VelocityWithGradient, as discussed in Section 3.1.
3.3.2 Kernel

The Kernel class is a templatable abstract base class for computing the Influence induced by an AeroElement or AeroAssembly at position $x$ in the flow. The superposition of all the individually induced Influences of AeroElements and the FreeStream yields the overall Influence at position $x$. With a sufficiently large collection of points, the entire flow field can be resolved, thus a Kernel that is templated to act on an AeroAssembly can be viewed as the equation that defines the induced flow field.

![Figure 3.7: Kernel<AeroElement> class hierarchy](image)

Notable descendants of Kernel with the AeroElement template parameter are VPGauss, VPVatistas, VFGarrel, VFVatistas, and VQFilamentized. Those beginning with VP act on VortexParticles; VF acts on VortexFilaments; and VQ acts on VortexQuads.

![Figure 3.8: Kernel<AeroAssembly> class hierarchy](image)

Notable descendants of Kernel when templated with AeroAssembly are Sequen-
tialKernel, BarnesHutKernel, and FastMultipoleKernel. SequentialKernel is the step-by-step $O(N^2)$ algorithm for superimposing individual AeroElement Influences. The BarnesHutKernel class employs the BHTC algorithm to reduce computational order to $O(N \log N)$, thus accelerating induced flow evaluations. FastMultipoleKernel improves on the BarnesHutKernel by precomputing lumped Influence approximations to deliver an approximately $O(N)$ algorithm. The FastMultipoleKernel class is an implementation of the FMM algorithm.

The BarnesHutKernel and FastMultipoleKernel rely on spatially sorting PF elements and computing summary data for groups of elements. At the same time, the numerical state of the PF elements is always evolving. Thus, summary values must be updated at a chosen interval in order to keep up with the changing numerical state. LibAero handles this by refreshing and rebuilding the tree object after a user-defined number of influence computation function calls. In this work, one-thousand calls was selected and used consistently throughout.

Other useful derivatives of the Kernel class are BilateralKernel and RotorKernel. The BilateralKernel class contains functionality for modelling bisymmetric flow fields using elements in only one-half of the field. A BilateralKernel object utilizes a normal vector and a point to define the reflection plane. The BilateralKernel class would be useful for modelling aircraft that have bilateral symmetry, although such work was not carried out in this thesis. The operation that defines the BilateralKernel is presented in Equation 3.2 where $I$ defines the total induced Influence and $i$ defines the partial Influence induced by the half of the wake structure that is modelled.

$$I(x) = i(x) + \text{reflect}(i(\text{reflect}(x)))$$  \hspace{1cm} (3.2)

Likewise, rotational symmetry is handled by the RotorKernel class, which requires an axis vector, an axis point, and the number of blades $N$. Equation 3.3 presents the defining equation of RotorKernel. Both BilateralKernel and RotorKernel are
templatable classes that can be applied to AeroElements and the AeroAssembly.

\[ I(x) = \sum_{k=0}^{N} \text{rotate}(i(\text{rotate}(x, 2\pi k/N), -2\pi k/N)) \]  \hspace{1cm} (3.3)

### 3.3.3 AeroEquation

The AeroEquation class is a templatable abstract base class for all equations that evolve AeroNodes, AeroElements, AeroAssemblies, and the AeroState. The template parameter can be any AeroObject such as Core. Here, the term *evolve* means to apply advection, diffusion, and deformation equations in order to update the numerical contents iteratively until convergence is reached. The AeroEquation base class was employed in order to give a common interface to evolving AeroObjects. Templating was employed so that the AeroEquation implementation has access to data that belongs to a specific type of AeroObject, while also providing type safety so that the compiler reports when the wrong object type is sent to an AeroEquation. In the case of steady-state or periodic simulations, the term evolution is the same as iterative convergence. For unsteady simulations, evolution should be viewed as the wake changing configuration step-by-step in time.

Advection, diffusion, and deformation models are classes that derive from AeroEquation with the template parameter set as Point, Core, and Strength respectively. Each AeroElement type requires its evolution model to be built as an array of advection, diffusion, and deformation models. Similarly, AeroAssembly evolution models, which derive from AeroEquation templated as AeroAssembly, are built as an array of AeroElement evolution models. Finally, the AeroState evolution model contains an array of AeroAssembly evolution models. This hierarchy forms the default basis of how equations are applied to updating the computational grid. Here, it becomes clear that keeping data and algorithms separate allows the programmer to break free of the default hierarchical scheme and create advanced algorithms that first reorganize the data for improved efficiency. In the Kernel category, FastMultipoleKernel is such an example. In the AeroEquation category of this section, however, the default hierarchical model is the only one pursued in this work.
Advection

The `AdvectE1F` class derives from `AeroEquation` with `AeroNode` as the template parameter. `AdvectE1F` is the first-order forward-Euler advection model, as indicated by the suffix `E1F`. The first-order forward-Euler equation is presented in Equation 3.4. `AdvectE2F` is the second-order forward-Euler advection model, and is presented in Equation 3.5.

\[
x_t = x_{t-\Delta t} + u_{t-\Delta t} \Delta t
\]

\[
x_t = x_{t-\Delta t} + u_{t-\Delta t} \Delta t + (u_{t-\Delta t} \cdot \nabla u_{t-\Delta t}) \frac{1}{2} \Delta t^2
\]

Going further, backward-Euler advection models exist, as shown in Equations 3.6 and 3.7. Backward equations have not found use thus far due to the posing of boundary conditions, which naturally suit explicit equation models and forward equations. This holds since wake elements emanate from the lifting line and are truncated at an arbitrary cutoff downwind. High-order equations such as the forward-backward Crank-Nicolson method can also be utilized, giving opportunity for future work.

\[
x_t = x_{t+\Delta t} - u_{t+\Delta t} \Delta t
\]

\[
x_t = x_{t+\Delta t} - u_{t+\Delta t} \Delta t - (u_{t+\Delta t} \cdot \nabla u_{t+\Delta t}) \frac{1}{2} \Delta t^2
\]
Diffusion

The first-order forward-Euler diffusion model is $\text{DiffuseE1F}$. The defining equation is given in Equation 3.8. The square-root occurs because the quantity that varies linearly is the variance $\sigma^2$ rather than the standard radius $\sigma$. The $\text{DiffuseE1F}$ class is not only a first-order model, but also the exact solution to the diffusion equation under the assumptions given in Section 2.1.2 where the diffusion equations were derived.

$$\sigma_t = \sqrt{\sigma_{t-\Delta t}^2 + 2(\nu + \nu_T)\Delta t}$$ \hspace{1cm} (3.8)

Deformation

Similarly, deformation models derive from $\text{AeroEquation}$ when templated as $\text{AeroNode}$. $\text{DeformE0F}$, the zero-order forward Euler equation, is given in Equation 3.9. This model is useful for PF elements such as $\text{VortexFilament}$ and $\text{VortexQuad}$. Due to conservation of circulation as described by Helmholtz, the vortex strengths of $\text{VortexFilament}$ and $\text{VortexQuad}$ elements must necessarily be equal to their vortex strengths in the previous snapshot, provided that division and merging do not occur. Furthermore, when a $\text{VortexFilament}$ or $\text{VortexQuad}$ lengthens, the core radius should decrease [30, 33]. The interaction between deformation and core size is not captured by the current models in $\text{LibAero}$.

$$s_t = s_{t-\Delta t}$$ \hspace{1cm} (3.9)

Unlike $\text{VortexFilament}$ and $\text{VortexQuad}$, which undergo strength deformation implicitly due to the separate advection of their endpoints and corner points, $\text{VortexParticles}$ require a strength deformation model that depends on velocity gradient to reorient and adjust the magnitude of its strength vector. Equation 3.10 is the first-order forward Euler strength deformation equation.

$$s_t = s_{t-\Delta t} + (s_{t-\Delta t} \cdot \nabla u)\Delta t$$ \hspace{1cm} (3.10)

3.4 Wing Data Classes

The $\text{WingObject}$ class is an abstract base class for all classes that hold numerical data about the state of wings and blades. At the start $\text{WingObjects}$ are initialized with
state data that represents the initial condition. Next, algorithms that derive from the \textit{WingEquation} class update the wing data, thus changing information such as the geometry and forces. The following subsections detail several implementation classes of \textit{WingObject} and explain how they are assembled to provide a numerical account of wings and rotors.

### 3.4.1 Wing

The \textit{Wing} class is a collection of numerical data that represents the current geometric configuration of one wing or one rotor blade. The \textit{Wing} class is built as an array of \textit{WingStations}. The \textit{WingStations} and can be laid out in any configuration, thus allowing \textit{Wings} to have arbitrary geometries. The coned, swept, and wingleted blades, as demonstrated in Chapter 5, rely on the capability to construct a wide range of blade and rotor geometries. Some consideration is required on the part of the user to ensure that the layout of \textit{WingStations} forms a feasible \textit{Wing} structure.
3.4.2 WingStation

The WingStation class contains data at a specific cross-section of the wing. The WingStation class comprises an array of WingNodes, which represent variables such as structural geometry, structural motion, local flow velocity and velocity gradient, angles-of-attack, and aerodynamic forces.

3.4.3 WingNode

WingNode objects are the data components contained in each WingStation. Derived classes of WingNode are SectionCoordinates, SectionState, SectionInfluence, AirfoilArguments, AirfoilCoefficients, and SectionForces. The rationale of dividing data into these categories is keeping like information together. This has the benefit that algorithms for updating quantities, say AirfoilCoefficients, can be interchanged more freely. The SectionCoordinates class contains the dimensionless spanwise coordinate $\mu$ of the particular WingStation. SectionState contains the quarter-chord position, velocity, and acceleration vectors as well as the chord vector and thickness vector of the airfoil at the particular WingStation. Structural models tie into the SectionState object. The structural motion model may be a simple rigid-body rotation model as used in this work for steady-state simulations or an advanced aeroelasticity model as recommended for future work.

SectionInfluence contains flow information. In current models, only the Influence at the quarter-chord position is stored. Advanced models that store Influence at several locations are foreseen, for example to account for dynamic pitching. The AirfoilArguments class contains angle-of-attack, Reynolds number, and thickness ratio. The calculation of AirfoilArguments relies on data that resides in SectionState and SectionInfluence. The AirfoilCoefficients class contains the lift coefficient, drag coefficient, and pitching moment coefficient. The final WingNode type is SectionForces, which comprises lift, drag, and pitching moment. Section 3.5 explains how data in WingNodes is utilized to update subsequent WingNodes.

3.4.4 WingAssembly

WingAssembly is a collection of Wings, which is useful for devices with more than one wing or blade. When symmetry models such as RotorKernel are employed, a multi-bladed turbine can be simulated using only one blade. Symmetry models apply only
when symmetry conditions hold, which is true in steady axial tower-free operation.

3.4.5 WingState

WingState is a collection of WingAssembly snapshots at different instants in time. As the topmost wing data class, the wing solver algorithm starts with WingState as its entry point and descends to lower levels and finally reaches WingNodes.

3.5 Wing Algorithm Classes

3.5.1 WingEquation

In order to evolve the numerical contents of the WingState, an update equation is applied hierarchically to the WingState, eventually reaching all WingNodes. The templatable parent class for equations that update wing information is WingEquation. The following sections present several subclasses of WingEquation with the various types of WingNode as the template parameter. Like WingObjects, WingEquations are also built into a hierarchy that corresponds to the WingObject hierarchy. For example, WingEquation with WingStation as the template parameter contains an
array of \textit{WingEquations} with \textit{WingNodes} as the template parameter. The hierarchy of algorithms continues with \textit{WingEquations} using \textit{Wing}, \textit{WingAssembly}, and \textit{WingState} as template parameters. The order of update starts at the first \textit{WingStation} of the first \textit{Wing}, then updates each \textit{WingNode}, continues on to the next \textit{WingStation}, and finally completes the update of every \textit{Wing}.

\section*{Preparatory Calculations}

In the steady-state analyses conveyed throughout this work, \textit{SectionCoordinates} are constant and the selection of stations along the wing or blade is known at the outset. However, it is possible to change the resolution of the blade by changing \( \mu \) values during the simulation. Structural models receive \textit{SectionState} as a function argument and update the geometry and motion of the wing. \textit{SteadyRotorMotion} is the simplest class of this type, and it computes the blade velocity and acceleration based on blade geometry information, but does not modify the geometry since the model is in steady-state. \textit{SteadyRotorMotion} inherits from \textit{WingEquation} with \textit{SectionState} as the template parameter. The \textit{SectionInfluenceUpdate} class updates \textit{SectionInfluence} by computing flow \textit{Influence} at the quarter-chord position of the wing section. \textit{SectionInfluenceUpdate} inherits from \textit{WingEquation} with \textit{SectionInfluence} as the template parameter.

\section*{Airfoil Calculations}

The \textit{AirfoilArgumentsUpdate} class computes the angle-of-attack, Reynolds number, and thickness ratio from the local flow and blade properties. These three quantities are utilized in airfoil coefficients interpolation tables. The look-up tables return lift coefficient \( C_L \), drag coefficient \( C_D \), and pitching moment coefficient \( C_M \). Figure 3.13 depicts a blade section alongside representations of blade velocity, flow velocity, lift force, drag force, and several intermediate quantities. Depicted intermediate quantities are \( u_c \), \( u_k \), and \( \alpha \). The relative velocity component in the edgewise direction is \( u_c \), and the relative velocity component in the flapwise direction is \( u_k \).

The airfoil arguments are computed according to Equations 3.11 through 3.16. Equations 3.11, 3.12, and 3.13 are used to precompute relative velocity components, particularly in the flapwise (thickness) direction, the edgewise (chord) direction, and in the plane of the airfoil respectively. Equations 3.14 and 3.15 compute the angle-of-attack and Reynolds number from the relative velocity components. Finally, Equation
3.16 gives a simple method of computing thickness ratio as a ratio of airfoil thickness to chord length. The thickness ratio is updated at every iteration; although superfluous under typical conditions, this methodology allows aeroelastic changes in wind geometry to feedback into the aerodynamics calculations. While dynamically adjusting the thickness ratio is possible since the current airfoil data tables include thickness ratio effects, dynamically adjusting the shape and hence coefficients curves of the airfoil would be challenging under the current framework.

\begin{align*}
    u_k &= \text{comp}(u_{\text{flow}} - u_{\text{blade}}, k) \quad (3.11) \\
    u_c &= \text{comp}(u_{\text{flow}} - u_{\text{blade}}, c) \quad (3.12) \\
    u_s &= \sqrt{u_k^2 + u_c^2} \quad (3.13) \\
    \alpha &= \arctan(u_k/u_c) \quad (3.14) \\
    Re &= \frac{|c|u_s}{\nu} \quad (3.15) \\
    Tr &= \frac{|k|}{|c|} \quad (3.16)
\end{align*}
Section Forces Calculation

Forces acting on the quarter-chord position of the airfoil section are computed from airfoil coefficients via Equations 3.17 through 3.20. Equation 3.17 is used to compute the lift force per blade length unit. Equation 3.18 is used to compute the drag force per blade length unit. Pitching moment per unit is computed using Equation 3.19. Finally, the total resultant force per length unit is computed as the sum of lift and drag components using Equation 3.20. Force calculations depend on fluid density, relative velocity, chord length, and airfoil orientation.

\[ L = \frac{1}{2} \rho u_s^2 |c| C_L \cdot \text{unit}(c \times k \times u_s) \] (3.17)

\[ D = \frac{1}{2} \rho u_s^2 |c| C_D \cdot \text{unit}(u_s) \] (3.18)

\[ M = \frac{1}{2} \rho u_s^2 |c| C_M \cdot \text{unit}(k \times c) \] (3.19)

\[ F = L + D \] (3.20)

3.6 Programming Pitfalls

This section presents the challenges in software organization that were encountered over the course of this work. These challenges fall into two main categories. The first category are challenges that relate to the nature of the numerical problem and are mostly independent of the programming language and paradigm choice. The choice of separating or integrating data and algorithms, as briefed in the introduction of Chapter 3, belong to the first category. The handling of shared nodes is another example. The second category are challenges that are typical for C++ programming, and are slightly exacerbated by the requirements or the particular approach of the numerical model. The requirement for deep-copying of data along with pointers to previous and following states as well as pointers to shared nodes belongs to the second category.

3.6.1 Separation of data and algorithms

The choice to separate data from algorithms, which was not anticipated at the outset, was made for two reasons that were realized during the course of the research. The first reason is simplicity and expressibility. In the integrated approach that is typical
of OOP, a viscous core model would contain not only its core radius, but also the algorithm to expand the radius and diffuse the core distribution. The core model could also contain the Biot-Savart induced flow model. The advantage of this approach is that every core model can have a unique diffusion equation. The disadvantage is that building such a complex system is an unusual requirement, and keeping a large number of core models is troublesome for code maintenance and reusability. This line of reasoning applies to advection, deformation, Biot-Savart equations, and wake evolution models, all which would expand to make a more complex code base.

The second reason for separating data and algorithms is crucial to the functioning of LibAero and the capability for future improvement of algorithms. Algorithms such as BHTC and FMM cannot function by traversing from element to element in the flow field. The BHTC and FMM algorithms must gather a list of PF elements and process the list to compute summary information such as a spatial tree. Although the wake evolution models rely on traversal, the traversal order must also be flexible, hence allowance must be made for listing and reorganizing objects. The rigid structure of each object containing its own algorithm does not facilitate smart algorithms of the type described. Additionally, proposed parallel algorithms rely on applying the same equation to arrays of values; this conflicts with the traversal methods that come along with integrating data and algorithms.

### 3.6.2 Shared Nodes

Another software organizational challenge was the handling of shared nodes. Vortex filament and quad elements have shared end- and corner-points respectively. Not only is the position shared, but also the flow influence at that position is shared between the elements. There are two popular approaches for handling data that is sometimes shared and other times owned exclusively. The first is to store only one chunk of data, and setup all included elements to point to the data chunk. If only one element is included, this means the data is exclusive to the element. The second approach is to store the data in each element exclusively. Additionally, each node can store a pointer to a second node if the first node follows the second. If the pointer value of a particular node is null, then the node is either an exclusive node or a master node. If the pointer value of this node is the address of another node, then this node is a slave node and the other node is its master node. In the first approach, the evolution algorithm updates each node and there are not any duplicates. In the

second approach, the evolution algorithm updates only master nodes, and sets the numerical values of slave nodes equal to the numerical values of the master node.

In early versions of LibAero, the first approach was employed. The main problem of this approach was the need for memory allocation and deallocation techniques to handle the creation and destruction of shared nodes. Memory-managed shared pointer classes were designed and implemented, the most successful being a hub-pointer model, which has a hub and observers and is similar to the master and slave model. The reason for the hub method is to ensure that the shared node is not updated twice or more. Unfortunately, the hub-pointer model cannot handle all possible allocation and deallocation schemes, such as merging and splitting PF elements. In fact, solvable problems emerged with deep-copying and having double deallocation errors that were sufficiently troublesome to question taking such a complex programming approach. Furthermore, the solution to such problems is to add more indicator and counter variables to the hub-pointer class, which brings into question one of the main purposes of sharing data chunks, memory savings.

Later versions of LibAero employed the second approach, wherein each shared node stores its position and influence as well as an optional pointer to a master node. Although this approach requires the advection equation to behave differently for a master node and for a slave node in addition to using more memory than the shared approach, it proved to be significantly more robust as a programming strategy. Modern programming languages, for instance JAVA, with built-in memory management techniques such as tracing garbage collection would facilitate the first approach. This would revive the debate between these two methods of handling shared nodes.

3.6.3 Deep Copying

One typical programming issue that was faced in this work was balancing the desire for deep-copying of numerical data with the necessity of pointers from one chunk of data to another. Deep-copying numerical data is useful because having the capability to copy a wake model with values and to use the new wake to seed a new simulation, or the capability to copy and rotate a wake model for graphing purposes, is a very convenient feature to have available. Chunks of data such as core models, called Cores, and shared points, called Points, contain pointers to other objects that represent the previous state, following state, or shared nodes. Clearly the the deep-copy method
should not produce clones of pointed objects such as the previous state, following state, or shared nodes. In addition, the new deep-copied object should not necessarily have pointers that point to the original pointed objects.

The deep-copy method of a wake model has two options and neither satisfies the intended aim. One option is to deep-copy a new object in the place of the object that is pointed; this does not accurately represent the wake model because it now has too many nodes or elements with the wrong pointer linkage. Another option is to copy only the pointer value; this does not accurately represent the wake model because the new wake now points into objects that belong to the old wake. The ideal outcome is for the new wake to have the exact configuration of the old wake and stand on its own.

The current solution for this problem is to avoid pointers altogether. Instead integer tags were employed to indicate the addresses of previous, following, and shared objects. In this approach, addresses are not memory addresses, but rather indexes to look up elements in arrays. In the most recent implementation, many integer tags are employed, including snapshot number, assembly number, wake number, element number, and node number. This method satisfies the requirement for deep-copying, but comes at the expense of a rigid hierarchy of snapshots, assemblies, wakes, elements, and nodes. Another expense is the complexity of using many integer addresses.

One proposed solution for the next version of LibAero is a hybrid of the pointer and the integer approach that is expected to satisfy the aim of deep-copying capability with the aims of simplicity and flexibility. In this approach, a single heap of all objects would be kept such that integer indexes can be used to access and copy any element. Another proposed solution is to change the requirement and dismiss the necessity for deep-copying. This is the simplest and most traditional approach taken by programmers. The drawback of this approach is the requirement for extra methods, such as a method to read the numerical values inside one wake model and write the values into another wake model of the same type.

3.6.4 Nested Object Arrays

Breaking data objects down into arrays of simpler objects is a design principle that was learned early in the project and has proven effective. The hierarchy of nodes, elements, wakes, and assemblies, in which each subsequent level is composed of an array of objects belonging to the lower level, is indispensable. The most recent version
Table 3.1: Pitfalls and achievements of *LibAero* development

<table>
<thead>
<tr>
<th>Version</th>
<th>Programming Language</th>
<th>Major Pitfall</th>
<th>Major Achievement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>MATLAB(^{TM})</td>
<td>Computationally very slow</td>
<td>Numerical success</td>
</tr>
<tr>
<td>0.2</td>
<td>C++</td>
<td>Rudimentary software architecture</td>
<td>Computationally fast</td>
</tr>
<tr>
<td>0.3</td>
<td>C++</td>
<td>Memory-managed node sharing</td>
<td>Advancement in OOP architecture</td>
</tr>
<tr>
<td>0.4</td>
<td>C++</td>
<td>Rigid object hierarchy</td>
<td>Robust node sharing</td>
</tr>
<tr>
<td>(0.5)</td>
<td>C++</td>
<td>(Unknown)</td>
<td>Flexible object hierarchy</td>
</tr>
</tbody>
</table>

of *LibAero* requires one prescribed hierarchy, which is too stringent, and introduced complications in the process of setting up periodic and unsteady simulations. In particular, the original strict hierarchy did not easily allow multiple wings or blades, and repairing this was not a trivial task after the architecture of connecting wings, wakes, and flow fields to one another as well as connecting past and future snapshots was established. The same difficulties were encountered with the prescribed structure of snapshots, leading to major challenges in constructing a periodic or unsteady simulation. Under the current rigid hierarchical system, periodic and unsteady simulations are still possible, but it is far from being a straightforward task.

Removing the requirement for a strict hierarchy, for example by using the heaped memory approach described previously, would strike the optimal balance between having a hierarchy of data object arrays and having flexibility with computational methods. Of course, the advantage of such a breakdown is the capability to descend into the hierarchy and find primitive objects such as *Points* to advect and *Cores* to diffuse, as well as gathering PF elements for computing the induced flow. Table 3.1 lists the pitfalls and achievements of the major versions of *LibAero* during the course of its software development.

### 3.7 Simulation Execution

This section explains the methodology of initializing, solving, and presenting simulation results. Initialization involves inputting airfoil curves and blade geometries, attaching wake models to blades, and selecting algorithm options. Solving requires relaxation iteration and convergence properties that are evaluated by an error measure. Presentation of results involves routines for plotting three-dimensional wake results with colour code for vortex strength. Other outputs include graphs of variables along the quarter-chord line of the blades such as angles-of-attack, Reynolds...
numbers, forces, and moments. Figure 3.14 presents a flow chart of the overall progression of setting up and running a LibAero simulation. The following paragraphs provide details of the process.

Setting up a simulation involves five steps. Choosing environmental conditions, particularly free stream wind speeds and directions, air density, kinematic viscosity, and turbulence parameters, is the first step. Selecting operating conditions such as rotor speed is the second step. The third step is building blades, each as an array of wing section geometries defined by a chord vector, thickness vector, airfoil shape, and airfoil coefficients table. Linear interpolation allows all geometric and airfoil properties to be accessed anywhere along the blade. The fourth step is to build the wake models, such as QuadParticleModel, and link each wake to the appropriate blade. The fifth step is to choose numerical models and settings such as advection, diffusion, deformation, and viscous core models. Step five also includes induction algorithms such as FMM and numerical settings such as the relaxation parameter.

Two main aspects are involved in understanding evolution of the simulation. First,
the current models of wake evolution step from primitive object to primitive object, updating each object once per iteration. Primitive objects are \textit{Point}, \textit{Core}, \textit{Strength}, and the \textit{LiftingLine}. Since the velocity must be computed at each \textit{Point} before advection and strength deformation are possible, the update of each \textit{Point} entails calling the induced flow algorithm. The second aspect to comprehend is the induced flow algorithm. Currently, the preferred induced flow algorithm is FMM, which does not sequentially sum the influence of PF elements. Rather, the BHTC and FMM algorithms organize the PF elements and precompute approximate summary information, thus accelerating flow field computations.

Figure 3.15 presents a flow chart that details the order of operations in the simulator, thus providing a detailed glimpse of the simulation block in Figure 3.14. At the start of the simulation routine, the first step is to initiate a feasible wake configuration in order to commence iteration. This is called the prescribed wake configuration. Blade forces can be estimated with the prescribed wake as a very fast and low order aerodynamics model. Wang and Coton employ such an approach [29]. When prescribing a rigid wake, improvements can be made by employing a flow field estimate that is different from the free stream, for instance using two-thirds of the free stream velocity.

The standard usage of \textit{LibAero} is to employ the free stream at the start to generate a feasible numerical state as a starting point for evolving the free wake. Between each iteration the new wake configuration is compared to the previous wake configuration. If they are sufficiently similar, the simulation is terminated. These differences are measured by computing the difference in flow velocity at each position node in the free wake from one iteration to the next. If the average flow velocity difference is less than a prescribed criterion such as 3\% or 5\% of the free stream velocity, then iteration is terminated and the current wake configuration is considered converged. Experience has shown that velocity differences are quite sensitive throughout the wake. As a result, mean velocity differences serve as a very conservative convergence test. Alternatively, angles-of-attack and blade forces can be utilized, which quickly converge to very small differences. In Chapter 4, setting the number of iterations to a fixed value such as ten was the preferred approach in order to make comparisons between other parameter settings without convergence issues impacting the results.

Two minor aspects fill in the details and make comprehension of the simulation algorithm clearer and more intuitive. These are the order of updating objects and the implementation of relaxation. The order of updating primitive objects has an impact
Setup

Iteration 0: prescribe the wake geometry by evolving the wake under the free stream flow field.

Output intermediates results.

Iteration i: evolve the free wake under the net flow field comprising the free stream and the induced flow field based on the free wake state of Iteration i-1.

Converged? Determine whether or not the mean difference in nodal velocities between iterations is less than 3% of the free stream velocity.

NO

YES

Postprocessing

Figure 3.15: Simulation flow chart

on computational performance and stability. Although the different possible orderings of update operations were not systematically studied due to scope of the work, it was discovered early in the project by experimentation that the update orders that yield better performance follow two rules of thumb. First, the order of update should start from the blade and move layer by layer to the far wake. Figure 3.16 presents the order-of-update for lifting lines and wakes that was employed throughout this work.
In the current implementation, the updated values of one object are utilized for updating the next object in the ordering. This is akin to Gauss-Seidel method for solving linear systems, and the opposite of Jacobi iteration which uses the only the numerical values of the previous iteration as a basis for updating the values of the current iteration. Second, and applicable only to periodic simulations, the algorithm should update the first element of each snapshot, then the second element of each snapshot, and so on. This requires the algorithm to cross-cut into the objects of different stages. This is necessary since objects in one stage depend on the numerical contents of objects in the previous stage.

Each individual evolution model, such as $\text{AdvectE1F}$ and $\text{DiffuseE1F}$, has its own relaxation parameter. In the current version of $\text{LibAero}$, when a particular relaxation setting is employed, the same relaxation setting is applied to all such models. The reason for retaining the option to employ disparate relaxation settings, and even to adaptively modify them, stems from the fundamental principle of relaxation iteration. Relaxation iteration entails strategically overshooting or undershooting the update estimate provided by the numerical algorithm. In the simple approach, all values in all parts of the numerical grid overshoot or undershoot by the same amount. The relaxation parameter is selected to balance over-stepping the correct solution in some areas of the computational grid and under-stepping the solution values in other areas.

Stability is another concern since over-stepping in some areas could lead to overall...
instability, suggesting a strategic bias towards under-relaxation. However, gradient-based methods recognize that certain types of values or regions of the numerical grid should target their updated values based on another parameter, namely gradient, in order to achieve optimal convergence performance. Although gradient-based methods are complex, possibly too complex for this problem, a more sophisticated relaxation method has been reserved for future work in order to accomplish the same goals.

The post-processing stage of a simulation involves the production of tabular and graphical results. Class methods are available to output tabular data. For instance, each blade model has methods to output angles-of-attack, Reynolds numbers, forces, moments, blade velocities, and flow velocities near the blade as a function of blade spanwise position. In addition, each wake model has a method to output tabular data of PF elements and their state values, which can be written to the screen or into a comma-separated variable (CSV) file. A post-processing routine converts the free wake data file into a Visualization Toolkit (VTK) graphics file, which can be viewed using a VTK viewer software such as Paraview.

Chapters 4 and 5 present numerous VTK-based rotor wake images as simulation outputs. Figure 3.17 presents a sample wake image in order to illustrate some of the key considerations in interpreting graphical outputs. This particular wake is a PrandtlSteadyParticleModel, which is shortened to the term Prandtl-Particle in some later descriptions. The wake is truncated after ten helical revolution of vortex elements, consisting of three revolutions of filaments and seven revolutions of particles. Only the wake of one blade is shown for the sake of simplicity, although this is the simulation output of a three-blade HAWT under steady axial inflow and rotor symmetry modelling. In this example, careful attention was paid to adjust the colour scheme and clearly present vortex monopole strength. Red indicates that the element has high vortex monopole strength, while blue indicate low strength.

Figure 3.17: Example rotor wake
The first feature to notice is the filaments along the blade have small diameter, while the filaments and particles increase in diameter moving downwind from the rotor; this is due to viscous diffusion. The second feature to notice is the cupped wake shape. The elements emanating from the tip region of the blade, and the hub region to some extent as well, experience faster flows and thus travel a greater distance in the same time. Here, an effect known as roll-up is very slightly visible; the outermost trailing helical filament from the tip of the blade curls inwards so that the new outermost filament is from the outboard region of the blade, but not from the tip.

The third feature to notice is the vortex strength pattern. The bound circulation was prescribed to follow an elliptical distribution along the blade while the blade elements are spaced according to a cosine distribution. As a result, the vortex filament trailing from the tip has low strength since the width of the bound filament that produced the trailing filament is very small. Nonetheless, the aggregate of vortex filaments trailing from the region near the tip have high vortex strength. After the transition from filaments to particles, particles obey the first-order strength equation. As a result, error accumulates in the particle strengths. This is noticeable as more red-coloured particles occur further downwind. A final feature to notice is the contraction of the wake in the downwind region. This is a side effect of wake truncation. The implication is that the wake must be sufficiently long such that the contracted region does not significantly affect numerical quantities in the rotor disc.
Chapter 4

Performance Evaluation

Chapter 4 provides evaluations of the numerical performance of *LibAero*. Section 4.1 employs numerical diagnostics to ensure that *LibAero* algorithms perform as expected and achieve numerically feasible results. The following sections focus on speed and accuracy tradeoffs of numerical model choices such as truncation, FMM parameters, and evolution models. Finally, Section 4.5 focuses on comparisons with experimental data from the MexNext Project and from the Tjæreborg wind turbine. Section 4.5 provides an evaluation of the agreement between *LibAero* simulations and real-world data sets for the sake of validating *LibAero* and determining the range of operating conditions that lead to acceptable simulation results.

Table 4.1: *LibAero* simulation settings

<table>
<thead>
<tr>
<th>Lifting Line / Wake Model</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Iterations</td>
<td>10</td>
</tr>
<tr>
<td>Relaxation Parameter</td>
<td>0.95</td>
</tr>
<tr>
<td>Truncation Revolutions</td>
<td>8</td>
</tr>
<tr>
<td>Number of Blade Elements</td>
<td>50</td>
</tr>
<tr>
<td>Blade Element Distribution</td>
<td>Cosine</td>
</tr>
<tr>
<td>Time Step (degrees)</td>
<td>15</td>
</tr>
<tr>
<td>Turbulent Viscosity Factor</td>
<td>443</td>
</tr>
<tr>
<td>Advection Diffusion</td>
<td>First-Order Forward Euler</td>
</tr>
<tr>
<td>FMM Cutoff Ratio</td>
<td>2</td>
</tr>
<tr>
<td>FMM Elements Per Leaf Node</td>
<td>4</td>
</tr>
</tbody>
</table>
LibAero simulation settings are laid out in Table 4.1. These simulation settings are employed as the default collection of settings throughout the remainder of the chapter. When a particular setting is varied, it is noted in that section of the text.

4.1 Numerical Diagnostics

This section investigates three numerical model parameters to ascertain their influence on the simulation results. For this purpose, the MexNext rotor is employed since experience and familiarity were established while conducting the validation analyses that are covered in Section 4.5.1. The rotation frequency is 424.5 rpm; this setting is used alongside the MexNext rotor throughout this work since a major portion of the MEXICO project experiments was performed at this setting. The circulation values along the blade were allowed to converge, rather than being prescribed as is typical for diagnostic procedures. The rationale is that it is more appropriate to perform diagnostics on realistic flow scenarios. The simulations in this section employ the PrandtlSteadyModel, composed entirely of vortex filaments.

The first numerical parameter is truncation distance. Truncation distance is measured in helical revolutions of free wake. As presented in Table 4.1, the default number of revolutions is eight whereas this section varies revolutions from one-half to thirty-two. The default settings listed in Table 4.1 apply through this chapter, except where an individual setting is varied for the sake of performing a parameter study. The second parameter covered is time discretization. Time discretization is measured in time units, but communicated as an angle. The angle represents the azimuthal displacement of a rotating blade in the duration of the time step. A time step of $15^\circ$ is most commonly used, although this section varies the time step from $3^\circ$ to $30^\circ$. The final parameter is number of iterations for convergence. The number was varied from zero iterations, known as the rigid or prescribed wake, to 20 iterations, with 10 being the default throughout this work.

4.1.1 Effects of Wake Truncation

A PF simulation is based on a discrete numerical model. As such the computational performance characteristics depend on discretization as well as truncation, since some of the boundary conditions are on the infinite horizon. For wind turbine wake simulations, the vortex wake trails toward infinity. The wake must somehow be terminated
to make the computational space finite. In this work, the wake elements are truncated after a specified lag time of their departure from the blade. Proposals are underway for the future development of a far wake model. Possibilities include modelling periodicity of the wake, appending semi-infinite vortex elements, and merging vortex elements downstream. Explanations are given in more detail in Chapter 6.

Truncation is measured in helical revolutions from the blade. This means there is not any cutoff plane located downwind of the rotor. Instead, the wake model prescribes a certain number of trailing element layers, for instance 96 layers. If the simulation time-step is set to correspond to a $15^\circ$ rotation of the rotor, then the rotor requires 24 time-steps to complete a full rotation. Since each layer corresponds to a time-step, 96 layers implies four revolutions in that time span. Therefore the wake will appear as approximately four helical revolutions. Tangential flow induction has a mild effect.

The method of truncation in terms of helical revolutions is justifiable. Because circulation is conserved, deformation effects cause faster moving parts of the flow to have more total vortex strength and exert more influence on the flow field. Thus it makes sense to truncate faster vortex trails further downstream. Truncation after a fixed number of layers does precisely that.

Figure 4.1 depicts three truncated wakes in their fully evolved forms. Figure 4.1a, on the left, is the shortest wake simulation; the wake is truncated after one-half of a helical revolution. Figure 4.1b, in the middle, is a medium length wake simulation; the wake is truncated after four helical revolutions. Figure 4.1c, on the right, is the longest wake simulation; the wake is truncated after 32 helical revolutions. Results show that four to eight helical revolutions of free wake provides an effective tradeoff between computational speed and accuracy.

Figure 4.2 presents the angle-of-attack curves along the blade for several trun-
cation levels ranging from 0.5 to 32 helical revolutions. With only four helical revolutions, the angle-of-attack curve closely matches the converged solution, which is considered to be the 32 helical revolutions model. Truncation error has a more pronounced influence on the inner portion of the blade as seen in Figure 4.2. Wake length has a large effect on axial induction and very little effect on tangential induction at the rotor disc, as shown in Figure 4.3. This, along with blade velocity considerations, accounts greater relative change in angles-of-attack on the inner portion of the blade.

Comparison of angle-of-attack curves is the most effective mode for comparing the effects of different numerical settings. Angle-of-attack and Reynolds number are the two components that determine the forces on the blade, given a specified blade geometry. Reynolds number depends mostly on rotor speed when tip-speed-ratio much greater than unity. As a result, angle-of-attack curves alone provide a clear comparison of effects across various numerical simulation settings.

The effects of truncation level on summary performance characteristics is presented in Table 4.2. The table confirms that eight helical revolutions provides simulation results that are sufficiently close the fully converged values.
Figure 4.3: Effect of truncation on flow induction

Table 4.2: Effect of truncation

<table>
<thead>
<tr>
<th>Helical Revolutions</th>
<th>$C_p$</th>
<th>$C_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.582</td>
<td>0.795</td>
</tr>
<tr>
<td>1</td>
<td>0.534</td>
<td>0.766</td>
</tr>
<tr>
<td>2</td>
<td>0.493</td>
<td>0.739</td>
</tr>
<tr>
<td>4</td>
<td>0.470</td>
<td>0.722</td>
</tr>
<tr>
<td>8</td>
<td>0.464</td>
<td>0.718</td>
</tr>
<tr>
<td>16</td>
<td>0.463</td>
<td>0.717</td>
</tr>
<tr>
<td>32</td>
<td>0.465</td>
<td>0.719</td>
</tr>
</tbody>
</table>

**4.1.2 Time Step Discretization**

Discretization in the context of PF simulation involves populating the wake model with a number of PF elements such that the application of the governing equations causes the numerical contents of the elements to evolve into an appropriately spaced and arranged configuration. For the wake models in this work, discretization refers to the number of filaments that compose the lifting line, the number of wake elements in the spanwise direction, time step duration, and the number of wake elements in the trailing direction. In this section, the effects of time step duration are explored. Time step is communicated as the rotation angle of the rotor in the time step duration. In this way, a clear image of the discretized arc lengths is conveyed.

Figure 4.4 depicts the fully converged wake in low and high temporal resolutions.
Figure 4.4: Depiction of temporal resolution

Figure 4.4a is the lowest resolution wake, having a time-step that corresponds to a $30^\circ$ angular displacement. Figure 4.4b is the highest resolution wake, having a time-step that corresponds to a $3^\circ$ angular displacement. The low resolution model appears jagged while the high resolution wake appears smooth.

Figure 4.5: Effect of temporal resolution on angles-of-attack

Using the angle-of-attack curve as the basis for comparison, Figure 4.5 reveals that within the range of $3^\circ$ to $30^\circ$ step sizes, choice of time-step does not have a major influence of angles-of-attack or forces. There is only a slight variation in the tip loss effect, as seen for $\mu$ between 0.9 and 1.0 where the angle-of-attack curves depart from
each other slightly. This makes sense from an intuitive standpoint. If a curved vortex filament is approximated as a straight filament, the induced flow is not substantially different, except in the region that is very close to the filament itself. Additionally, the viscous core model ensures that the induced flow near the filament is small. As a result, the flow field is robust to changes in time step. In the tip region, the vortex filaments strengths are the strongest, leading to larger discrepancies. Table 4.3 verifies that there is very little change in power and thrust coefficients for the selected time steps.

Table 4.3: Effect of temporal resolution

<table>
<thead>
<tr>
<th>Time Step</th>
<th>$C_p$</th>
<th>$C_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$30^\circ$</td>
<td>0.454</td>
<td>0.711</td>
</tr>
<tr>
<td>$15^\circ$</td>
<td>0.455</td>
<td>0.712</td>
</tr>
<tr>
<td>$6^\circ$</td>
<td>0.459</td>
<td>0.715</td>
</tr>
<tr>
<td>$3^\circ$</td>
<td>0.454</td>
<td>0.710</td>
</tr>
</tbody>
</table>

4.1.3 Blade Section Discretization

This section analyzes the effect of changing the number of blade sections. Figure 4.6 presents the angle-of-attack curves for several blade discretization schemes. All discretization schemes involve blade sections that are situated along the blade according to a cosine distribution. The number of blade sections in order of increasing resolution are 5, 10, 20, and 40. Figure 4.6 indicates that convergence is achieved with approximately 40 blade sections, since the angle-of-attack curves of 20 sections and 40 sections are very similar. Table 4.4 corroborates this result by presenting the power and thrust coefficients at the various discretization levels.

Table 4.4: Effect of bladed discretization

<table>
<thead>
<tr>
<th>Number of Blade Elements</th>
<th>$C_p$</th>
<th>$C_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.451</td>
<td>0.688</td>
</tr>
<tr>
<td>10</td>
<td>0.455</td>
<td>0.710</td>
</tr>
<tr>
<td>20</td>
<td>0.448</td>
<td>0.705</td>
</tr>
<tr>
<td>40</td>
<td>0.449</td>
<td>0.697</td>
</tr>
</tbody>
</table>
4.1.4 Convergence Behaviour

This section looks at convergence behaviour with the aim of determining the number of free wake iterations that are necessary to achieve a fully evolved solution. Figure 4.7 presents angle-of-attack curves, the standard mode of comparison. The graph presents the zero iteration case, which is known as the prescribed or guessed wake configuration. The prescribed wake is a simple helical structure generated from the free stream velocity field without any rotor induction. After induction effects from vortex elements are included in later iteration steps, the helical wake evolves into a helical cup-like shape that depends on the rotor design and operating conditions. The first, second, and third iteration results are given, as well as the twentieth iteration result.

Figure 4.7 suggests that there is little difference between the simulation results after three iterations. Generally, five iterations is sufficient for convergence. Table 4.5 verifies that convergence is rapid for power and thrust results as well. In this specific instance, four and five iterations provided the best results with a slight increase in the error measure with a larger number of iterations. However, ten iterations is a more reasonable default number for analyses where ensuring convergence is necessary. In the case of design space exploration when speedy simulation runs are required, five iterations would serve as a better rule of thumb.
Figure 4.7: Effect of iteration number on angles-of-attack

Table 4.5: Effect of iteration number

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$C_p$</th>
<th>$C_t$</th>
<th>Error Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.788</td>
<td>0.902</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.436</td>
<td>0.701</td>
<td>4.85491</td>
</tr>
<tr>
<td>2</td>
<td>0.382</td>
<td>0.660</td>
<td>1.6846</td>
</tr>
<tr>
<td>3</td>
<td>0.461</td>
<td>0.717</td>
<td>1.26803</td>
</tr>
<tr>
<td>4</td>
<td>0.451</td>
<td>0.712</td>
<td>1.08056</td>
</tr>
<tr>
<td>5</td>
<td>0.454</td>
<td>0.713</td>
<td>0.88777</td>
</tr>
<tr>
<td>10</td>
<td>0.457</td>
<td>0.716</td>
<td>1.12796</td>
</tr>
<tr>
<td>20</td>
<td>0.457</td>
<td>0.716</td>
<td>1.08758</td>
</tr>
</tbody>
</table>

4.2 Computational Acceleration

This section provides an evaluation of numerical speed and accuracy of the methods that *LibAero* employs for accelerating computations. The three methods covered in this section are relaxation, BHTC, and FMM. Relaxation is the method of strategically overshooting or undershooting the target values of the next iteration in order to accelerate convergence and improve numerical stability. The section examines the numerical performance of several relaxation levels. The following two methods, BHTC and FMM, are two implementations of the tree-code algorithm for sorting PF ele-
ments spatially in order to accelerate flow field computations. The section examines the effect of varying the cutoff parameter, which determines the aggressiveness of the tree-code approximation.

4.2.1 Relaxation Method

The relaxation method was applied to the solver in order to achieve faster convergence in solving the equation system. In this work, the relaxation parameter was either unity, which is standard for a fixed-point iterative method, or less than unity, known both as under-relaxation and smoothing. Figure 4.8 demonstrates the influence that smoothing has on convergence speed. The x-axis of the graph is the iteration number and the y-axis is the measure of convergence.

The specific convergence measure used here is the average absolute difference between flow velocities of the previous iteration and the new unrelaxed flow velocity estimates. The set of velocities includes all Points in the bound lifting line and free wake. A residual difference of approximately 1.0 m/s is noticeable. The residual difference is due to rearrangement of nodal positions in the far regions of the wake. Although a difference of 1.0 m/s is significant, the difference is comparatively small considering the free stream velocity is 15 m/s. Meanwhile, it is important to note that this particular convergence measure was selected because it provides a conservative convergence test when compared with angles-of-attack or blade forces. The entire velocity field was chosen as a convergence criterion in order to be as strict as possible and achieve convergence to the fullest level possible.

Figure 4.8 reveals that a small amount of smoothing accelerates convergence substantially. A low smoothing parameter has a dampening effect on oscillations that develop due to the nonlinearity inherent to the advection term in the vorticity transport equation, given in Equation 2.14. Figure 4.9 depicts angle-of-attack curves, which makes the differences recognizable in order to be sure that smoothing does not introduce substantial error. The figure indicates that a smoothing parameter of 0.8 leads to a small amount of error when compared to the standard solution with a relaxation parameter of 1.0. As the number of iterations increases beyond five or ten, the error in the velocity field increases very slowly, most likely due to floating point error. Further work is necessary to investigate the nonlinear phenomenon that leads to blade variables converging apparently to definite values while the free wake geometry does not fully converge in a feasible time frame.
Table 4.6 demonstrates that performance coefficients begin to experience error when smoothing is lowered to 0.8. This compliments the results given in Figure 4.9. A smoothing parameter of 0.90 to 0.95 provides substantial improvement in convergence speed while maintaining accuracy. This work uses a relaxation setting
of 0.95 throughout.

Table 4.6: Effect of relaxation setting

<table>
<thead>
<tr>
<th>Relaxation</th>
<th>( C_p )</th>
<th>( C_t )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>0.454</td>
<td>0.711</td>
</tr>
<tr>
<td>0.95</td>
<td>0.458</td>
<td>0.713</td>
</tr>
<tr>
<td>0.90</td>
<td>0.457</td>
<td>0.712</td>
</tr>
<tr>
<td>0.80</td>
<td>0.452</td>
<td>0.708</td>
</tr>
</tbody>
</table>

4.2.2 Barnes-Hut Tree Code

BHTC and FMM are two related methods that accelerate induced flow field computations using lumped approximations. As detailed in Section 2.5, both methods rely on a tree data structure to sort PF elements that are nearby from those that are far away from each other. Far-away elements are lumped together and approximated with multipoles. In this work the simple monopole is employed, which is equivalent to a point vortex at the centre of the cluster of lumped elements. The cutoff setting for deciding when to utilize the far-away approximation determines the aggressiveness of the approximation. A cutoff setting of 0.25 is very aggressive; this setting indicates that the ratio of the distance between the point in question and the edge of the bounding sphere that surrounds all the elements in the tree node to the diameter of the bounding sphere must be greater than 0.25. On the other extreme, a cutoff distance of 4.0 is very conservative, producing results that are very close to those of the direct algorithm. This section and the following section, which cover BHTC and FMM respectively, continue to employ the vortex filament wake model.

Figure 4.10 presents angle-of-attack curves with the BHTC cutoff setting ranging from 0.25 to 4.0, and the figure provides the direct algorithm as well for comparison purposes. With the cutoff set at 1.0, the angle-of-attack curve is reasonably accurate. At the 2.0 setting, the angle-of-attack curve is very accurate. Since the purpose of the approximation procedure is computational speed, the comparison to determine the preferred cutoff setting should consider computational speed as well as accuracy.

Table 4.7 shows how the performance coefficients compare to the direct algorithm. In addition, the table shows the simulation run times produced by each BHTC cutoff setting and the percent of time saved when compared to the direct algorithm. A cutoff setting of 2.0, while being accurate, only saves 26% of the run time. The cutoff
Figure 4.10: Effect of BHTC cutoff setting on angles-of-attack

Table 4.7: Effect of BHTC cutoff setting

<table>
<thead>
<tr>
<th>Cutoff</th>
<th>(C_p)</th>
<th>(C_t)</th>
<th>Time (s)</th>
<th>Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>0.480</td>
<td>0.728</td>
<td>127</td>
<td>87%</td>
</tr>
<tr>
<td>0.5</td>
<td>0.470</td>
<td>0.721</td>
<td>218</td>
<td>78%</td>
</tr>
<tr>
<td>1</td>
<td>0.463</td>
<td>0.717</td>
<td>408</td>
<td>59%</td>
</tr>
<tr>
<td>2</td>
<td>0.458</td>
<td>0.713</td>
<td>743</td>
<td>26%</td>
</tr>
<tr>
<td>4</td>
<td>0.451</td>
<td>0.708</td>
<td>1086</td>
<td>-8%</td>
</tr>
<tr>
<td>Direct</td>
<td>0.455</td>
<td>0.711</td>
<td>1002</td>
<td>0%</td>
</tr>
</tbody>
</table>

setting of 1.0 saves 59% and offers a good balance between computational speed and accuracy. Cutoff settings of 0.5 and 0.25 save 78% and 87% respectively, suggesting a possible role in very aggressive approximations. Future work should focus on refined approximation strategies such as high fidelity near the blade and low fidelity in the far wake or low fidelity in early iterations and high fidelity in later iterations.

4.2.3 Fast Multipole Method

Advancements on BHTC led to the development of FMM, which further accelerates computations by employing precomputation of far-away lumped flow influences. Figure 4.11 demonstrates the accuracy of FMM. Like BHTC, FMM maintains suitable
accuracy when the cutoff setting is 1.0, and is very accurate when the cutoff is set to 2.0.

![Graph showing effect of FMM cutoff setting on angles-of-attack](image)

**Figure 4.11: Effect of FMM cutoff setting on angles-of-attack**

Table 4.8 presents the performance coefficients and computational time results. The table reveals that FMM saves more time than BHTC for similar error levels, thus FMM is generally preferred. The setting of 2.0 saves 43% in the FMM case compared to 26% for BHTC. A cutoff of 0.5 saves 88% on run time while maintaining reasonable accuracy for angles-of-attack and performance coefficients. This makes FMM at the 0.5 cutoff setting an excellent choice when the aim is speedy simulation.

**Table 4.8: Effect of FMM cutoff setting**

<table>
<thead>
<tr>
<th>Cutoff</th>
<th>$C_p$</th>
<th>$C_t$</th>
<th>Run Time (s)</th>
<th>Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>0.474</td>
<td>0.723</td>
<td>66</td>
<td>94%</td>
</tr>
<tr>
<td>0.5</td>
<td>0.468</td>
<td>0.720</td>
<td>129</td>
<td>88%</td>
</tr>
<tr>
<td>1</td>
<td>0.460</td>
<td>0.715</td>
<td>282</td>
<td>73%</td>
</tr>
<tr>
<td>2</td>
<td>0.455</td>
<td>0.711</td>
<td>597</td>
<td>43%</td>
</tr>
<tr>
<td>4</td>
<td>0.453</td>
<td>0.710</td>
<td>984</td>
<td>7%</td>
</tr>
<tr>
<td>Direct</td>
<td>0.455</td>
<td>0.711</td>
<td>1055</td>
<td>0%</td>
</tr>
</tbody>
</table>

Table 4.9 presents the impact of PF element selection on the computational performance of FMM. The columns of the table indicate the number of each type of
element in the wake model as well as the computational time results. The wake models that predominantly consist of vortex particles posted greater computation speed enhancement with FMM. The filament-particle model and quad-particle model both provided 90% time savings. This compares to 66% and 82% for the pure filament and pure quad models respectively. Particle elements have a positive interaction with the tree-code algorithm because the bounding radii of particles are smaller, leading to smaller bounding radii of the tree nodes that contain particles. Smaller bounding radii allow the cutoff method to include more elements in the far-away approximation without substantially increasing error. The choice to model the wake with particle elements, which are of lower order in element geometry, may introduce error in the first place, however.

Table 4.9: Effect of element choice on FMM

<table>
<thead>
<tr>
<th>Wake Model</th>
<th>Particles</th>
<th>Filaments</th>
<th>Quads</th>
<th>Direct Time (s)</th>
<th>FMM Time (s)</th>
<th>Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prandtl</td>
<td>0</td>
<td>4678</td>
<td>0</td>
<td>5846</td>
<td>2013</td>
<td>65.6%</td>
</tr>
<tr>
<td>PrandtlParticle</td>
<td>3456</td>
<td>454</td>
<td>0</td>
<td>5792</td>
<td>579</td>
<td>90.0%</td>
</tr>
<tr>
<td>Quad</td>
<td>0</td>
<td>70</td>
<td>4416</td>
<td>84284</td>
<td>14965</td>
<td>82.2%</td>
</tr>
<tr>
<td>QuadParticle</td>
<td>3312</td>
<td>70</td>
<td>1104</td>
<td>28858</td>
<td>2858</td>
<td>90.1%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Wake Model</th>
<th>Direct $C_p$</th>
<th>Direct $C_t$</th>
<th>FMM $C_p$</th>
<th>FMM $C_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prandtl</td>
<td>0.462</td>
<td>0.719</td>
<td>0.468</td>
<td>0.724</td>
</tr>
<tr>
<td>PrandtlParticle</td>
<td>0.454</td>
<td>0.714</td>
<td>0.469</td>
<td>0.724</td>
</tr>
<tr>
<td>Quad</td>
<td>0.511</td>
<td>0.754</td>
<td>0.48</td>
<td>0.731</td>
</tr>
<tr>
<td>QuadParticle</td>
<td>0.531</td>
<td>0.765</td>
<td>0.493</td>
<td>0.74</td>
</tr>
</tbody>
</table>

4.3 Induction and Evolution Equations

This section examines the impact that changing induction and evolution models has on LibAero simulation results. The induction models, known as Kernels, are the result of the viscous core distribution of a vortex element. The first subsection investigates the effects of four viscous core model choices for a vortex filament wake model. The second subsection examines advection models: the first-order advection model relies on flow velocity, the second-order model includes flow acceleration, and another
experimental model that attempts to account for turbulent wake effects is shown for the sake of completeness. The final subsection investigates the effect of varying the turbulent viscosity parameter, thus changing the diffusivity of vorticity.

4.3.1 Viscous Core Modelling

This section provides an analysis of the effects of Biot-Savart Kernels for diffuse PF element geometries. Figure 4.12 compares four vortex filament viscous core models. The models are Scully (also known as Vatistas-1), Vatistas-2, Rankine (Vatistas-∞), and Garrel. The Biot-Savart equations that define these core models were presented in Sections 2.3.3 and 2.3.2. The differences between the angle-of-attack results of different core models are noticeable. The maximum difference is approximately a one degree difference in the tip loss region, and occurs between the Rankine model and the Vatistas-2 model. Over most of the angle-of-attack curve, the models maintain agreement within 0.2 degrees of each other. The cause of the differences between viscous core models depends mostly on the fatness of the tails of the core distribution. The Garrel core has high kurtosis, implying fat tails, thus axial induction at the rotor plane is slightly reduced and angles-of-attack are higher. Another mode of difference between simulation results is the effect that core models have on inducing flow at nearby neighbouring elements, thus influencing the converged wake configuration.

![Figure 4.12: Effect of viscous core models on angles-of-attack](image-url)
Table 4.10 presents the performance coefficients that result from PrandtlSteady-Model simulations employing different vortex filament core models. The Garrel core and Vatistas-2 core give higher $C_p$ and $C_t$, which agrees with the increased angles-of-attack in Figure 4.12. The Vatistas-2 core produces increased angles-of-attack only in the tip region of the blade, which is the most influential region in determining $C_p$ and $C_t$. The range in $C_p$ results is approximately 5% and the range in $C_t$ results is approximately 2%. Thus a moderate amount of wake model tuning can be carried out by strategic selection of viscous core models. Future work is recommended to show consistent accuracy in one core model over others as an indication that the preferred core model provides a better physical description of the real viscous core fluid effects.

<table>
<thead>
<tr>
<th>Kernel</th>
<th>$C_p$</th>
<th>$C_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scully</td>
<td>0.456</td>
<td>0.712</td>
</tr>
<tr>
<td>Vatistas-2</td>
<td>0.476</td>
<td>0.725</td>
</tr>
<tr>
<td>Rankine</td>
<td>0.452</td>
<td>0.710</td>
</tr>
<tr>
<td>Garrel</td>
<td>0.472</td>
<td>0.723</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>0.464</td>
<td>0.717</td>
</tr>
</tbody>
</table>

### 4.3.2 Advection Models

The choice of advection model is another key ingredient in the system of governing equations. The first-order forward-Euler model $AdvectE1F$ and second-order forward Euler model $AdvectE2F$ are compared in this section. Another model, based on the first-order forward-Euler, with the addition of a random Gaussian term, proposed to account for turbulent mixing and large-scale unsteadiness was evaluated. This model is called $AdvectE1FTurb$.

Figure 4.13 presents the angle-of-attack curves that result from each advection model. The graph reveals that the experimental model that includes randomized turbulent displacements has very little influence on the end result. The second-order model, however, is noticeably different from the first-order model in angles-of-attack, amounting to a difference of a quarter of a degree roughly. The difference in angles-of-attack is attributed to high flow accelerations in the blade tip region, which influence the geometry of the converged free wake.
Table 4.11: Effect of advection models

<table>
<thead>
<tr>
<th>Advection Model</th>
<th>Cp</th>
<th>Ct</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Order</td>
<td>0.462</td>
<td>0.716</td>
</tr>
<tr>
<td>Second Order</td>
<td>0.485</td>
<td>0.733</td>
</tr>
<tr>
<td>First Order Turbulent</td>
<td>0.462</td>
<td>0.717</td>
</tr>
</tbody>
</table>

Table 4.11 presents performance coefficients of the selected advection models. It reveals the second-order model causes a noticeable increase in $C_p$ and $C_t$, which agrees with the increase in angles-of-attack. The effect warrants further investigation since the first-order model outperformed the second-order model in matching the MEXICO experimental results where $C_p$ was 0.437 and $C_t$ was 0.687, which are covered in detail in Section 4.5.1.

### 4.3.3 Diffusion Models

This section explores the influence that prescribed turbulent viscosity has on steady HAWT rotor wake simulation results via changing the core diffusivity. Figure 4.14 shows the angle-of-attack curves when the turbulent viscosity factor is set to 0, 100, and 10000 respectively. The wide range of values was chosen due to lack of literature on the proper choice of turbulent viscosity for rotor wake flows and because
experimentation led to the conclusion that over a wide range of turbulent viscosity values there is little variation in simulation results. In order to estimate a reasonable target range of turbulent viscosity values, figures in Leishman [3] were interpreted, indicating that the range of values from 50 to 500 were most appropriate. The differences in numerical results are quite small overall, indicating very small dependence of the aerodynamics results on the guessed value of turbulent viscosity. Even with zero turbulent viscosity and hence near-zero diffusivity, the core radius remains at the initial value, which is the thickness of the blade. Therefore, singularity and stability issues are mitigated, even without considering diffusion under turbulence. A turbulent viscosity factor of 10000 smooths out the angles-of-attack in the tip region and leads to minor increase $C_p$ and $C_t$ values, as presented in Table 4.12.

Figure 4.14: Effect of turbulent viscosity on angles-of-attack

<table>
<thead>
<tr>
<th>Turbulent Viscosity</th>
<th>$C_p$</th>
<th>$C_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.460</td>
<td>0.715</td>
</tr>
<tr>
<td>100</td>
<td>0.462</td>
<td>0.716</td>
</tr>
<tr>
<td>10000</td>
<td>0.469</td>
<td>0.721</td>
</tr>
</tbody>
</table>
4.4 Wake Models and Settings

4.4.1 Wake Models

The wake models described in Chapter 3 are constructed from the available pool of PF elements. Figure 4.15 depicts the wake models that are evaluated in this section. The top row of the figure is devoted to wake models where vortex filaments trail from the rotor blade for a chosen distance, before transitioning into vortex particles. In all cases where it is not otherwise specified, transitioning into vortex particles involves the conversion of one filament into one particle or one quad into one particle. The following subsection, however, performs a small parameter study of transitioning one filament or quad into more than one particle element.

Figure 4.15a on the left presents a wake model composed entirely of vortex filaments, spanning eight helical revolutions. The top centre image presents a wake model with four revolutions of filaments and four revolutions of particles. The top right image has two revolutions of filaments and six revolutions of particles. The bottom row is similar to the top row, except that vortex filaments are replaced with vortex quads while the particle configuration remains the same as in the image directly above.

Figure 4.15 demonstrates the main characteristics of PF wake modelling. The helical structure due to blade rotation is the most obvious feature. The PF elements trail downwind and grow larger due to diffusion. Diffusion is most noticeable by...
observing how vortex particles become darker further downwind. The radii of the filaments and particles are drawn at 10% of the actual Gaussian standard radius of the PF element in order to show more detail in the wake structure without overlapping elements cluttering the scene. Wake expansion is visible near the rotor disc. Far downstream the wake contracts; this is a numerical side effect of wake truncation.

![Figure 4.16: Effect of wake model on angles-of-attack](image)

Figure 4.16: Effect of wake model on angles-of-attack

Figure 4.16 presents the angle-of-attack curves of the six wake models. While the overall shape is similar for the different models, there are noticeable differences. Models built with vortex quads bring about higher angles-of-attack. Likewise, the more particles in the wake model, the higher angles-of-attack become. Higher angles-of-attack imply higher $C_p$ and $C_t$; this effect is verified in Table 4.13.

<table>
<thead>
<tr>
<th>Wake Model</th>
<th>$C_p$</th>
<th>$C_t$</th>
<th>Run Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filaments 8 / Particles 0</td>
<td>0.468</td>
<td>0.724</td>
<td>2179</td>
</tr>
<tr>
<td>Filaments 4 / Particles 4</td>
<td>0.469</td>
<td>0.724</td>
<td>8797</td>
</tr>
<tr>
<td>Filaments 2 / Particles 6</td>
<td>0.469</td>
<td>0.724</td>
<td>2635</td>
</tr>
<tr>
<td>Quads 8 / Particles 0</td>
<td>0.480</td>
<td>0.731</td>
<td>8380</td>
</tr>
<tr>
<td>Quads 4 / Particles 4</td>
<td>0.484</td>
<td>0.734</td>
<td>9362</td>
</tr>
<tr>
<td>Quads 2 / Particles 6</td>
<td>0.493</td>
<td>0.740</td>
<td>6625</td>
</tr>
</tbody>
</table>
Table 4.13 presents simulation run times in addition to summary performance coefficients. Wake models that are based on vortex quads have longer run times due to the numerical integration required by vortex quad *Kernels*. In contrast to results in Sections 4.2.2 and 4.2.3, the models without vortex particles have shorter run times because they employ the first-order advection model that does not require velocity gradient computations. Only vortex particles require velocity gradient for their strength deformation model. Finally, wake models that use a large number of vortex particles are fast, since particle elements enhance the speed gains of FMM; this agrees with earlier results.

### 4.4.2 Vortex Particle Transition and Spacing

Another wake modelling option is conversion of vortex filaments or vortex quads into more than one vortex particle when transitioning to the far wake. Thus the final vortex filament in the trail can be split into two or more vortex particles. Likewise, the final vortex quad is split into $M \times N$ particles. Figures 4.17a and 4.17b depict such wake models. Figure 4.17a on the left shows each filament transitioning into four particles, initially in a line, then evolving separately thereafter. Figure 4.17b on the right shows each quad transitioning into four particles in a two by two arrangement, initially planar and evolving separately.

![Vortex particle transition](image)

(a) Filament $\rightarrow$ particles (4)  
(b) Quad $\rightarrow$ particles ($2 \times 2$)

Figure 4.17: Vortex particle transition

Figure 4.18 compares vortex particle arrangements. The graph suggests that increasing the number of particles leads to a slight increase in angles-of-attack, although this effect is very subtle and difficult to filter from natural variations in converged solutions. Table 4.14 presents performance coefficients and simulation times. The
higher angles-of-attack appear as higher $C_p$ and $C_t$ in Table 4.14.

<table>
<thead>
<tr>
<th>Wake Model</th>
<th>$C_p$</th>
<th>$C_t$</th>
<th>Run Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Filament → 1 Particle</td>
<td>0.471</td>
<td>0.726</td>
<td>579</td>
</tr>
<tr>
<td>1 Filament → 2 Particles</td>
<td>0.471</td>
<td>0.726</td>
<td>1202</td>
</tr>
<tr>
<td>1 Filament → 4 Particles</td>
<td>0.478</td>
<td>0.730</td>
<td>1830</td>
</tr>
<tr>
<td>1 Quad → 1 × 1 Particles</td>
<td>0.507</td>
<td>0.748</td>
<td>1429</td>
</tr>
<tr>
<td>1 Quad → 2 × 2 Particles</td>
<td>0.510</td>
<td>0.751</td>
<td>3328</td>
</tr>
</tbody>
</table>

4.5 Experimental Validation

This section covers the experimental validation of LibAero. The first portion of this section presents comparisons to the MEXICO Experiment, which makes up part of the MexNext Project. Comparisons between blades forces, power, thrust, and velocimetry are given. Additionally, BEM simulation results are presented alongside the LibAero results to show how LibAero compares to another simulation tool. The BEM simulation tool that was chosen was ExcelBEM, a MatLab$^{TM}$ and C-based code...
developed by Crawford to handle various scenarios including coned HAWT rotors [5]. The second portion of this section offers a brief comparison of *LibAero* simulation results to the experimental power curves of the Tjæreborg wind turbine.

### 4.5.1 MEXICO Experiment

This section presents a validation analysis of the *LibAero* simulator against the MEXICO HAWT rotor aerodynamics experiment under steady axial wind conditions. The MEXICO experiment is a part of the MexNext Project, a collaboration of worldwide researchers to aggregate experimental wind turbine data for validation purposes. Member research groups regularly contribute validations and analyses. The MEXICO experiment covers a range of steady and unsteady aerodynamics experiments; a large part of the steady-state portion is covered in this work.

The experimental wind speeds were 10 m/s, 15 m/s, and 24 m/s. In the work presented in this section, the angular velocity of the rotor was 424.5 rpm in all cases. Figure 4.19a depicts the MexNext wind turbine as situated in the wind tunnel. Figure 4.19b presents a diagram of the rotor blade, which is composed of three different airfoils, a root cylinder, and blended regions. In particular, the three airfoils are DU-91-W2-250, Risø-121, and NACA-64418. This analysis relies on interpolation tables of sectional airfoil coefficients as well as interpolation of neighbouring airfoils. The airfoil coefficients tables were gathered as part of the MEXICO experiment [42].

The *ExcelBEM* simulation settings are a combination of four factor settings: hub loss model, expanding wake model, centrifugal pumping model, turbulent mixing momentum correction model. Hub loss, expanding wake, and centrifugal pumping have two switch settings, on and off. Turbulent wake has three settings: Bladed\textsuperscript{TM}, Glauert, and Spera. This brings a total of twenty-four sets of simulation settings for *ExcelBEM*.

Three sets of *ExcelBEM* settings labelled *Low*, *Best*, and *High* are communicated in the following results. The name *Low* was chosen because this model generally had the lowest forces and power; *High* has the highest forces and power; and *Best* was the model that fit the experimental data best overall. Table 4.15 presents the settings that are referred by the labels *Low*, *Best*, and *High*. Further details on BEM modelling of the MexNext HAWT, specifically using *ExcelBEM*, are provided by Cline [43].
(a) Wind turbine  
(b) Rotor blade

Figure 4.19: MexNext wind rotor configuration

Table 4.15: ExcelBEM simulation settings

<table>
<thead>
<tr>
<th>Model</th>
<th>Hub Loss</th>
<th>Expanding Wake</th>
<th>Centrifugal Pumping</th>
<th>Turbulent Wake</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>On</td>
<td>On</td>
<td>Off</td>
<td>Glauert</td>
</tr>
<tr>
<td>Best</td>
<td>On</td>
<td>Off</td>
<td>Off</td>
<td>Glauert</td>
</tr>
<tr>
<td>High</td>
<td>Off</td>
<td>On</td>
<td>On</td>
<td>Spera</td>
</tr>
</tbody>
</table>

Analysis of Blade Forces

Figure 4.20 presents six graphs that provide a summary comparison of local forces along the blade. The comparison includes the MEXICO experimental data, ExcelBEM simulation results under three combinations of settings, and the results of four LibAero wake models. The MEXICO experimental blade force data was collected by integrating pressure curves, while the pressure curves were gathered using 25 to 28 pressure sensors at each of five blade stations.

Airfoil normal force graphs are presented in the left-hand side of Figure 4.20. Positive normal force is defined as the component of the total force pointing in the flapwise direction from the lower (pressure) surface to the upper (suction) surface of the airfoil. Normal force can alternatively be defined as the magnitude of the total force that is perpendicular to the edgewise direction. However, this method does not
resolve sign, which is necessary here. Airfoil tangential force graphs are presented in the right-hand side. Positive tangential force is defined as the component of the total local aerodynamic force that points in the edgewise (chordwise) direction from leading edge to trailing edge. Normal and tangential force graphs are presented for three cases: 10 m/s, 15 m/s, and 24 m/s steady wind speeds in axial (zero-yaw) operation. The graphs of 10 m/s, 15 m/s, and 24 m/s cases are layered at top, middle, and bottom respectively.

Figures 4.20a and 4.20b present the airfoil forces under 10 m/s steady wind conditions. This case implies a tip speed ratio of 10.0 and high axial induction. Since there are only five experimental data points, it is difficult to make a definitive comparison. Both LibAero and ExcelBEM methods overpredict forces slightly. The graphs indicate that LibAero models and ExcelBEM models are similarly predictive. Later discussion of $C_p$ and $C_t$ will shed more light on the comparison since integrated forces bring about power and thrust.

The 15 m/s case with tip speed ratio of 6.7, shown in Figures 4.20c and 4.20d, indicates that the LibAero and BEM models again overpredict slightly and are similarly predictive to one another. Here, the tip speed ratio is 6.7, close to the typical optimal performance tip speed for HAWT rotors. For this reason, conformance to the 15 m/s case can be regarded as a higher priority. A pattern emerges showing that the QuadParticle wake model has the greatest error among all simulation models.

The 24 m/s case with low axial induction and a tip speed ratio of 4.4 is presented in Figures 4.20e and 4.20f. The experimental data points appear erratic, suggesting unsteadiness or possible measurement error. The LibAero models and BEM models largely agree with one another, especially given the unique shape of the tangential force curve with the dip in the middle. Altogether, the predictive agreement of LibAero models with the MexNext experimental airfoil force results appears reasonable, but not perfect.

Analysis of Power and Thrust

Table 4.16 presents a global results comparison of the MEXICO experiment, ExcelBEM simulations, and LibAero simulations. In particular, $C_p$ and $C_t$ are presented as global summary results in contrast to the force curves presented for the comparison of local results. Coefficients are tabulated at each wind speed level, and an overall error measure is given to compare simulation effectiveness. The experimental $C_p$ and
Figure 4.20: MEXICO blade section forces
$C_t$ values were computed by numerical integration of the five force measurements along the blade. This suggests that the experimental $C_p$ and $C_t$ values are not precise measurements.

Table 4.16: MEXICO power and thrust results comparison

<table>
<thead>
<tr>
<th></th>
<th>10 m/s $C_p$</th>
<th>10 m/s $C_t$</th>
<th>15 m/s $C_p$</th>
<th>15 m/s $C_t$</th>
<th>24 m/s $C_p$</th>
<th>24 m/s $C_t$</th>
<th>Average Absolute Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>MexNext</td>
<td>0.299</td>
<td>0.861</td>
<td>0.437</td>
<td>0.687</td>
<td>0.227</td>
<td>0.385</td>
<td></td>
</tr>
<tr>
<td>BEM (low)</td>
<td>0.286</td>
<td>1.021</td>
<td>0.430</td>
<td>0.798</td>
<td>0.202</td>
<td>0.392</td>
<td>-4.3% 18.6% -1.6% 16.2% -11.0% 1.8% 8.9%</td>
</tr>
<tr>
<td>BEM (best)</td>
<td>0.298</td>
<td>1.026</td>
<td>0.430</td>
<td>0.798</td>
<td>0.207</td>
<td>0.393</td>
<td>-0.3% 19.2% -1.6% 16.2% -8.8% 2.1% 8.0%</td>
</tr>
<tr>
<td>BEM (high)</td>
<td>0.447</td>
<td>1.162</td>
<td>0.497</td>
<td>0.851</td>
<td>0.233</td>
<td>0.417</td>
<td>49.5% 35.0% 13.7% 23.9% 2.6% 8.3% 22.2%</td>
</tr>
<tr>
<td>Prandtl</td>
<td>0.381</td>
<td>1.044</td>
<td>0.467</td>
<td>0.818</td>
<td>0.207</td>
<td>0.393</td>
<td>27.3% 21.3% 6.8% 19.1% -8.8% 2.0% 14.2%</td>
</tr>
<tr>
<td>Prandtl-Particle</td>
<td>0.385</td>
<td>1.063</td>
<td>0.453</td>
<td>0.808</td>
<td>0.207</td>
<td>0.393</td>
<td>28.9% 23.5% 3.7% 17.6% -8.6% 2.1% 14.1%</td>
</tr>
<tr>
<td>Quad</td>
<td>0.316</td>
<td>1.047</td>
<td>0.485</td>
<td>0.832</td>
<td>0.206</td>
<td>0.392</td>
<td>5.6% 21.6% 11.0% 21.1% -9.3% 1.9% 11.8%</td>
</tr>
<tr>
<td>Quad-Particle</td>
<td>0.476</td>
<td>1.183</td>
<td>0.510</td>
<td>0.849</td>
<td>0.203</td>
<td>0.391</td>
<td>59.2% 37.4% 16.7% 23.6% -10.6% 1.6% 24.8%</td>
</tr>
</tbody>
</table>

Table 4.16 results indicate that ExcelBEM models were slightly better than LibAero models for predicting $C_p$ and $C_t$. The LibAero models generally overpredicted $C_p$ and $C_t$, just as they overpredicted forces along the blade, especially in the 10 m/s wind speed case where the axial induction is high. The cause of error is not known with certainty, but turbulent mixing effects are proposed as the most likely cause. LibAero does not adequately account for turbulent mixing and the associated unsteady effects whereas the semi-empirical Glauert turbulent wake model employed by ExcelBEM takes unsteady turbulent effects into account. This comes about in the Glauert model where the relationship between thrust coefficient and axial induction factor is modelled as a theoretical parabola based on momentum theory for low inductions and modelled with linear regression of experimental data for high induction factors. To illustrate, the 10 m/s case gave an estimated axial induction factor of 49.8%, while
the 15 m/s case corresponds to 24.8% and the 24 m/s case corresponds to 11.2%. The latter two cases agree reasonably well and the axial induction is less than 30%, the typical changeover point for turbulent wake modelling.

Additionally, it is difficult to explain the inaccuracy of the QuadParticle model compared to the PrandtlParticle model, given that quadrilateral elements provide higher fidelity than filaments due to their improved resolution in the spanwise direction due to linear variation of circulation. Improved accuracy was attained with the pure Quad model. The average absolute error in coefficients was 8.0% for the best ExcelBEM model, and it was 11.8% for the LibAero quad model.

Analysis of BEM Results

This section carries on with the same comparison that was presented in the previous section. However, these comparisons of quarter-chord axial induction and tangential induction are directed at LibAero and BEM, but do not warrant as much emphasis on the MEXICO aerodynamics data. Although MEXICO experimental induction factors computed from particle image velocimetry (PIV) data are included when possible, the data is insufficiently precise for making definitive conclusions. For instance, the induction factor estimates in this section involve averaging PIV velocity data points that were located 10° before and after the blade. The azimuthal averaging of PIV points can only serve as a rough estimate of the flow behaviour near the blade since the technique smears out and mostly eliminates the tip loss effects.

Figure 4.21 presents the axial induction factors for the three cases: 10 m/s, 15 m/s, and 24 m/s wind speeds. In 10 m/s high induction case of Figure 4.21a, the PIV data corroborates BEM and LibAero models to a large extent, with the Glauert-model BEM result agreeing very closely. The Quad-based LibAero models underpredict axial induction, which contributes to the higher $C_p$ and $C_t$ values, as detailed previously. Although not as pronounced in the 15 m/s case, Figure 4.21b suggests that the filament-based LibAero models provide a slightly better fit. Finally, the 24 m/s low induction case in Figure 4.21c shows good agreement between BEM and LibAero models, while the MEXICO PIV data points are too different from the simulation results for meaningful comparison, as well as being slightly noisy in data quality.

Figure 4.22 presents tangential induction factors along the blade for the three simulation cases. The graphs reveal that ExcelBEM and LibAero are in good agreement, except at the root and tip regions in some cases. The general trend is that
Figure 4.21: MEXICO axial induction
Figure 4.22: MEXICO tangential induction
tangential induction factors from \textit{LibAero} models are slightly higher than those from \textit{ExcelBEM} models. The tangential induction factors estimated from PIV are noisy, hence difficult for comparison.

Figure 4.23 presents the angle-of-attack curves of \textit{ExcelBEM} models and \textit{LibAero} models in 10 m/s, 15 m/s, and 24 m/s wind regimes. The results reflect most of the characteristics that were noted previously. The \textit{LibAero} and \textit{ExcelBEM} curves largely agree. The 24 m/s low induction case presented in Figure 4.23c indicates remarkable agreement between the angle-of-attack curves of all \textit{LibAero} and \textit{ExcelBEM} models. In other cases where differentiation occurs, the \textit{LibAero QuadParticle} model appears to have the most error. In addition, the \textit{LibAero} filament models produce a sharp oscillation in angle-of-attack near the tip, as well as a smaller oscillation near the root in the 10 m/s high induction case. Sharp changes in angle-of-attack are due to sharp changes in flow velocity, which owe to the filament model concentrating vorticity to a filament region, rather than smoothing the rotational effects along a sheet as done with quad elements.

\textbf{Analysis of Flow Field Velocimetry}

The MEXICO experiment acquired particle image velocimetry (PIV) data for each set of operating conditions by taking measurement traverses and gathering position-velocity pairs. The traverses used in this work were radial traverses. Some were on the rotor disc, while others were downstream by up to 2.63 meters, filling a 120° slice of the cylindrical region with data points. These traverses are sweeps from the hub toward the circumference of the rotor disc. Traverses were made at angles of -90°, -70°, -50°, -30°, -10°, 10°, and 30°. The negative angles are behind the blade in azimuth and the positive angles of ahead. [42]

Each traverse acquired approximately 7000 data points. Each traverse of data was acquired in repetition of at least ten times and averaged so as to reduce measurement error. Even so, there is a high degree of variability between neighbouring data points. A smoothing algorithm was developed to lump nearby data points together and estimate the flow velocity field with minimal error. The smoothing algorithm searched the data space in increments of 0.1 meters and included points within a radius of 0.05 meters, using the centroid as the centre point. This algorithm reduced the number of position-velocity pairs to approximately 3175, depending slightly on the particular data set (10 m/s, 15 m/s, or 24 m/s wind cases).
Figure 4.23: MEXICO angles-of-attack
Figure 4.24 is a chart of the flow field velocities predicted by the *LibAero* filament wake model plotted against the MEXICO experimental PIV velocities at the same collocation points. The red dots represent the x-component of velocity; this is the axial component. The blue dots are the y-component, pointing in the spanwise direction of the primary blade. The green dots are the z-components of flow velocity, pointing in the direction rotated 90° forward from the primary blade.

There is a high degree of agreement between the experimental data and the simulated results. This is apparent since the scattered points follow the diagonal line in Figure 4.24. However, there are observations that are quite far from the diagonal as well. Further investigation should be conducted to discover how much of the scatter is due to systematic simulation error and how much belongs to random measurement error.

![Figure 4.24](image)

(a) High Induction (10 m/s)  
(b) Medium Induction (15 m/s)

**Figure 4.24**: Comparison with particle image velocimetry results

Table 4.17 presents a simple approach for removing random experimental errors and focusing on systematic simulation error. The table gives a percentage measure of the relative error between simulated flow velocities and experimental flow velocities from PIV measurements. The basis of the relative error is the free stream velocity. The table indicates that errors in the y- and z-components are smaller and not obviously systematic. Average error in axial velocity is larger, ranging from 3.5% to 5.0%. In each case the *LibAero* simulation overpredicts axial velocity. In other words, *LibAero* underpredicts axial induction factors, since axial induction points in the opposite direction of the free stream velocity.
Table 4.17: MEXICO velocimetry vs. LibAero flow field

<table>
<thead>
<tr>
<th></th>
<th>10 m/s</th>
<th>15 m/s</th>
<th>24 m/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-component (axial)</td>
<td>3.83%</td>
<td>4.87%</td>
<td>3.54%</td>
</tr>
<tr>
<td>y-component</td>
<td>2.36%</td>
<td>-0.47%</td>
<td>-0.85%</td>
</tr>
<tr>
<td>z-component</td>
<td>-1.70%</td>
<td>-0.29%</td>
<td>-0.34%</td>
</tr>
</tbody>
</table>

4.5.2 Tjæreborg Wind Turbine

The Tjæreborg wind turbine is a 2 MW HAWT stationed in Tjæreborg, Denmark, which has made available experimental power curves. The turbine is three-bladed and has a diameter of 61 meters including 12 meters of hub. It is a constant-speed turbine with a rotational speed of 22.1 RPM. The blade geometry follows a simple scheme with linear chord variation and linear twist variation. The root chord is 3.3 meters and the tip chord is 0.9 meters. The twist varies 1° per three meters. Furthermore, the blade sections consist of NACA-44XX airfoils [44].

![Figure 4.25: Tjæreborg power and thrust](image)

In order to validate LibAero, the filament-based wake model called PrandtlModel was employed to simulate the Tjæreborg wind turbine. The results of the simulation are presented in Figure 4.25. Figure 4.25a, on the left, presents the simulated power coefficient alongside the experimental power coefficients across various wind speeds. Figure 4.25b, on the right, presents the simulated thrust coefficient. Figure 4.25a indicates that LibAero provides an adequate description of the power curve. At low wind speeds, LibAero underpredicts $C_p$. For medium and high wind speeds, LibAero overpredicts $C_p$. The error keeps within plus or minus 10%, except for the three
highest wind speeds where the errors in $C_p$ predicted by $LibAero$ climb to 14%, 25% and 35% respectively.

In contrast to the MexNext validation performed in Section 4.5.1, agreement between $LibAero$ and the Tjæreborg experimental data is strongest for low wind speeds. Figure 4.25a indicates that agreement is satisfactory for wind speeds under 14 m/s, while the best agreement is in the optimal performance range of approximately 8 m/s wind to 10 m/s wind. Future work is recommended to explain the discrepancy at high wind speeds where the tip speed ratio and the axial induction factor are lower.
Chapter 5

Applications

In Chapter 5, demonstrations of PF aerodynamics simulations are presented. The objective is to demonstrate the variety of simulations that can be achieved under the current software methodology of LibAero. Suggestions are made on possible design improvements and lines of future work that would lead LibAero to a greater degree of simulation options.

Four simulation scenarios are presented in the following sections.

1. A three-bladed coned HAWT rotor in steady axial flow.
2. A three-bladed swept HAWT rotor in steady axial flow.
3. A two-bladed HAWT rotor with winglets in steady axial flow.
4. A three-bladed standard Danish HAWT rotor in yawed flow.

In each section, an analysis is provided to suggest how aerodynamic, structural, and economic benefits could be achieved through non-standard wind rotor designs. In all cases, the twist distribution is not optimal, thus results are preliminary and for demonstration purposes. Default simulation settings are laid out in Table 5.1. These settings are fairly standard throughout the chapter; when a particular parameter is varied, it is noted in writing in that section. The yawed flow case involves a periodic simulation and employs entirely different settings.
Table 5.1: Default simulation settings

<table>
<thead>
<tr>
<th>Blade Modified MexNext</th>
</tr>
</thead>
<tbody>
<tr>
<td>Angular Velocity (rpm)</td>
</tr>
<tr>
<td>Lifting Line / Wake Model</td>
</tr>
<tr>
<td>Relaxation Parameter</td>
</tr>
<tr>
<td>Number of Iterations</td>
</tr>
<tr>
<td>Truncation Revolutions</td>
</tr>
<tr>
<td>Number of Blade Elements</td>
</tr>
<tr>
<td>Blade Element Distribution</td>
</tr>
<tr>
<td>Time Step (degrees)</td>
</tr>
<tr>
<td>Turbulent Viscosity Factor</td>
</tr>
<tr>
<td>Advection Diffusion</td>
</tr>
<tr>
<td>Vortex Particle Deformation</td>
</tr>
<tr>
<td>Vortex Filament Deformation</td>
</tr>
<tr>
<td>Vortex Quad Deformation</td>
</tr>
<tr>
<td>FMM Cutoff Ratio</td>
</tr>
<tr>
<td>FMM Elements Per Leaf Node</td>
</tr>
</tbody>
</table>

5.1 Steady Axial Flow Simulations

5.1.1 Coned Wind Rotor

Coned HAWT rotors have been a design option since the mid-1990s [45]. However, passive and active coning mechanisms add cost and complexity in addition to existing mechanisms for pitch and yaw control [5]. In order for coning to prove as valuable as pitch and yaw, substantial cost and performance benefits that offset the added costs and complexity should be realized first. Thus far the proposed benefits have not achieved that level as recognized by the wind power industry.

Research work carried out by Crawford [5] and predecessors [46, 47, 48] has shed light on the major concerns of coned rotors. Significant direct aerodynamic performance increases are not expected due to the reduction of frontal area for energy capture caused by coning. However, the blades can be made longer, and the structural benefits of load shedding have the potential to increase aerodynamic performance. Coning for load shedding in high winds and protection from extreme winds appears most promising. Coning offers the possibility of capturing power at full generator capacity in high winds. Coning for protection offers the possibility of more complete protection from extreme winds by reducing wind capture area. Pitching does not
protect against wind gusts and yaw does not protect against sudden changes in wind direction, whereas reducing capture area more universally mitigates adverse wind conditions. The ultimate objective is a reduction in blade strength and rotor costs that could be achieved with coned rotors [5].

In this section, a coned HAWT rotor is analyzed in five coning states: -30°, -15°, 0°, 15°, and 30°. Negative angles represent upwind coning and positive angles are downwind coning. The default settings in Table 5.1 are employed; the lifting line and wake model is set to \textit{PrandtlSteadyModel}, which is composed entirely of vortex filaments. This analysis is concerned with aerodynamic comparisons, rather than load shedding and protection, since the emphasis of this work is steady-state aerodynamics. Figures 5.1a and 5.1b depict 15° upwind and 15° downwind coning states respectively. The blade design that is employed for the coning rotor analysis is the MexNext rotor blade, which was detailed in Subsection 4.5.1 and depicted in Figure 4.19b. Having amassed data and understanding of the MexNext rotor, it serves as a convenient reference point for numerical studies such as the coned rotor study pursued in this section.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{coning_rotors.png}
\caption{Coned rotors}
\end{figure}

The mechanics of coning are simple: each blade is rotated about its root location using the appropriate rotational axis for coning. In this way the blades are not distorted numerically. The most suggestive simulation results are the forces along the blade as presented in Figures 5.2a and 5.2b. Force results give an indication of the cases where blades can be made lighter and cheaper, thus saving on costs. Figure 5.2a details the axial forces along the blade from root to tip. Axial forces are the
components that contribute to thrust and thus influence the structural strength of the tower. Figure 5.2b presents the rotational forces along the blade. Rotational forces are the components that contribute to electric generator torque and power output.

![Graphs showing axial and rotational forces](image)

Figure 5.2: Coned rotor loadings

Figures 5.2a and 5.2b indicate that coning forward or backward reduces axial and rotational forces along the blade because coning reduces the frontal energy capture area. Under closer examination, the figures show that forward coning reduces forces on the inner portions of the blade, whereas backward coning reduces forces on the outer portions of the blade. As such backward coning offers more potential for cost reduction since forces on outer portions of the blade contribute more to bending moments due to cantilevering. Even when the blade root bending moments (BRBM)s of upwind and downwind coning match one another, at least the outer portion of the blade can benefit from a reduced structural design in the case of downwind coning. In other words, bending moments along the outer portion of the blade are reduced in the backward coning case. This effect is noticeable in Figures 5.2a and 5.2b by comparing the $-15^\circ$ coning case to the $+15^\circ$ coning case and $-30^\circ$ coning to $+30^\circ$ coning. In all cases, the crossover point where the force on the forward coned blade surpasses the force on the backward coned blade is at approximately $\mu = 0.7$.

Table 5.2 presents summary performance values for the MexNext rotor in selected coned configurations. The $C_p$ and $C_t$ columns indicate a slight increase in power coefficient and thrust coefficient as the rotor is coned from upwind to downwind. The $C_p$ and $C_t$ figures are computed on a basis of the frontal area with the reduction in frontal area due to coning included. From a practical point of view, it would not be
Table 5.2: Effect of coning angle

<table>
<thead>
<tr>
<th>Coning (deg)</th>
<th>λ</th>
<th>$C_p$</th>
<th>$C_t$</th>
<th>Power (W)</th>
<th>Thrust (N)</th>
<th>BRBM (N·m)</th>
<th>$C_p/C_t$</th>
<th>Power/BRBM (W/N·m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-30</td>
<td>5.78</td>
<td>0.463</td>
<td>0.704</td>
<td>11540</td>
<td>1168</td>
<td>561.0</td>
<td>0.659</td>
<td>20.6</td>
</tr>
<tr>
<td>-15</td>
<td>6.44</td>
<td>0.463</td>
<td>0.712</td>
<td>13978</td>
<td>1431</td>
<td>679.7</td>
<td>0.651</td>
<td>20.6</td>
</tr>
<tr>
<td>0</td>
<td>6.67</td>
<td>0.465</td>
<td>0.718</td>
<td>14899</td>
<td>1536</td>
<td>720.2</td>
<td>0.647</td>
<td>20.7</td>
</tr>
<tr>
<td>15</td>
<td>6.44</td>
<td>0.471</td>
<td>0.725</td>
<td>14222</td>
<td>1459</td>
<td>680.2</td>
<td>0.650</td>
<td>20.9</td>
</tr>
<tr>
<td>30</td>
<td>5.78</td>
<td>0.478</td>
<td>0.729</td>
<td>11916</td>
<td>1211</td>
<td>560.1</td>
<td>0.656</td>
<td>21.3</td>
</tr>
</tbody>
</table>

Feasible to cone both upwind and downwind with the same rotor due to the tower positioning of conventionally designed HAWTs. A wind turbine would be designed from outset as an upwind coning device with a downwind tower or a downwind coning device with an upwind tower. According to the analysis conducted in this section, the downwind coning HAWT with upwind tower is the preferred design due to the load shedding properties mentioned previously. Other advantages of downwind coning not considered in the present models are yaw stability and the potential for passive yaw control. Since this section focuses on a simplified tower-free aerodynamic analysis, the HAWT rotor is free to be coned upwind and downwind as desired.

Power and thrust decrease as the coning angle increases due to the reduction in frontal area. Looking more closely at the table reveals that power per thrust $C_p/C_t$ benefits slightly from coning upwind or downwind. This suggests that with coning the same power could be delivered with a lighter, cheaper tower. The effect is very small, however, and the dominant concern still appears to be blade design and costs. The ratio of rotor power to BRBM in the far right column of Table 5.2 suggests potential performance benefits from coning downwind. Again the effect appears small compared to the frontal area loss and cost of a coning mechanism as expected from a steady-state aerodynamic analysis. These differences do, however, shed light on the general trends in forces and moments suggesting possible benefits in extreme wind scenarios where the effects are larger.

### 5.1.2 Swept-Bladed Wind Rotor

Like coned rotors, swept-bladed rotors have been under design consideration for decades. Most consideration is in the helicopter community where higher Mach numbers justify swept blades for the reduction of transonic effects, which are not present
for the low speeds of wind turbines [3]. Unlike coned rotors, typical swept blades do not reduce frontal area substantially, though the blade length does increase slightly due to curvature. This depends to a large extent on the particular implementation of sweep.

Swept blades have the inherent disadvantage of complicating structural requirements. Pitching moment becomes a larger issue for blades when the aerodynamic centre is non-linear. Furthermore, centrifugal forces and inertial coupling concerns are exacerbated [3]. As a result, most curvilinear blade designs are not fully swept, but instead have a raked wingtip region. Examples of swept-bladed HAWTs are the small wind turbines of SkyStream\textsuperscript{TM} [49] and the STAR research HAWT rotor of Sandia National Laboratories [50]. SkyStream claims aeroelastic and structural improvements due to tip sweeping. Despite the obstacles, swept blades offer potential aerodynamic efficiency benefits by taking advantage of azimuthal flow field variations as well as offering aeroelastic and load shedding improvements.

Figure 5.3 depicts backward and forward swept rotors accompanied by a \textit{Quad-SteadyModel} wake, which is composed of a vortex filament lifting line and a free wake of vortex quadrilateral sheets. The swept rotor employed in this work is based on the MexNext rotor, but with the quarter-chord line of the blade bowed into the shape of a parabola. A sweep parameter of +20% indicates the mid-span quarter-chord position is bowed in the direction of rotation by 20% of the blade length; this is called backward sweep. A negative sweep parameter means the blade bows opposite of its rotation direction; this is called forward sweep. The basis for the sweep measurement is a straight blade, defined as having 0% sweep. Across all sweep cases, the tip radii
and the frontal areas are constant.

The standard coordinates used throughout this work indicate that the rotor rotates counter-clockwise when viewing the rotor from the positive x-axis or downwind side. Since the view presented in Figure 5.3 is an upwind perspective, the rotor rotates clockwise. This information is intended to assist the reader in understanding the orientations and the sweep definition. It is also useful to notice the shape of the airfoils in the figure in order to understand the direction of blade rotation.

Figure 5.4 presents the quarter-chord curves at three sweep settings. The shapes are plotted on a relative scale, having the blade length normalized to unity. The purpose of Figure 5.4 is to demonstrate the parabolic shape of the quarter-chord curve, which is not clearly depicted in Figure 5.3 alone. Like 5.3, Figure 5.4 is properly scaled so the curvature of the quarter-chord curve accurately represents the actual bow of the swept blade.

Figures 5.5a and 5.5b display the forces along the blade for several swept-bladed HAWT configurations. Axial forces shown in Figure 5.5a and rotational forces shown in Figure 5.5b indicate that force per meter along the blade is decreased on outboard portions of the blade for both forward and backward sweep. A portion of the reduction in force per meter is accounted for by the curvature of the blade, since the blade bends most in the root and tip areas in relation to the on-coming wind that results from
the relative velocity between the flow field and the moving blade.

Table 5.3: Effect of blade sweep

<table>
<thead>
<tr>
<th>Sweep (%)</th>
<th>$C_p$</th>
<th>$C_t$</th>
<th>Power (W)</th>
<th>Thrust (N)</th>
<th>$C_p/C_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-30%</td>
<td>0.428</td>
<td>0.793</td>
<td>13731</td>
<td>1696</td>
<td>0.540</td>
</tr>
<tr>
<td>-20%</td>
<td>0.432</td>
<td>0.784</td>
<td>13861</td>
<td>1675</td>
<td>0.552</td>
</tr>
<tr>
<td>-10%</td>
<td>0.453</td>
<td>0.800</td>
<td>14544</td>
<td>1710</td>
<td>0.567</td>
</tr>
<tr>
<td>0%</td>
<td>0.499</td>
<td>0.840</td>
<td>16006</td>
<td>1795</td>
<td>0.594</td>
</tr>
<tr>
<td>10%</td>
<td>0.529</td>
<td>0.859</td>
<td>16968</td>
<td>1836</td>
<td>0.616</td>
</tr>
<tr>
<td>20%</td>
<td>0.545</td>
<td>0.863</td>
<td>17480</td>
<td>1845</td>
<td>0.632</td>
</tr>
<tr>
<td>30%</td>
<td>0.545</td>
<td>0.885</td>
<td>17475</td>
<td>1892</td>
<td>0.616</td>
</tr>
</tbody>
</table>

Table 5.3 presents summary results for the tested configurations of swept-bladed HAWTs. Power, thrust, $C_p$, $C_t$, and $C_p/C_t$ all increase as sweep moves backward. The latter suggests potential structural cost savings, especially on tower requirements, by utilizing back-swept blades. A detailed analysis of power, thrust, and bending moments is required to determine the optimal amount of backward sweep, which would ideally be carried out alongside an optimized twist distribution.

As mentioned previously, the improved aerodynamic performance is attributed to swept blades taking advantage of azimuthal variation in the flow field. However, this serves only as a partial explanation. It may be possible in some cases to redesign the chord and twist distributions of a straight blade to match the angles-of-attack and
Reynolds numbers achieved with the swept blade. In that case the performance of the swept blade would be matched by the straight blade. Because this is not a full optimization analysis, the amount of efficiency improvement that can be captured by redesigning the straight bladed rotor remains unknown. Figure 5.6 presents the higher angles-of-attack that are achieved with the backward swept blades.

5.1.3 Wind Turbine Rotor with Winglets

Winglets are wingtip devices that consist of a small wing section that is attached to the end of a standard wing with some joint angle. In specialized designs, particularly for aircraft wings, a winglet may be blended into the wing itself, instead of a designing a sudden change in the quarter-chord orientation. Additionally, there are raked winglets that qualify as swept blades in addition to being winglets [3]. The simple winglet employed in this analysis is attached to the rotor blade at 90° (or 0° cant angle) and points in the upwind direction. Presumably, it is attached to an upwind HAWT rotor so as to avoid tower collision. This is in contrast to the more usual design where winglets are placed on the suction side, which would be downwind for HAWTs. The toe angle of the winglet is a measure of twist angle for winglets and was varied in this trial. Toe angles of -6°, 3°, 0°, 3°, and 6° are explored. Positive toe angle is defined by the leading edge of the winglet being a slightly greater distance from the hub than the
trailing edge of the winglet; this arrangement induces inward radial flow. A negative toe angle is defined as the opposite and induces outward radial flow.

Figure 5.7: Two-bladed rotor with winglets

Figure 5.7 displays the geometry of the rotor employed in this exploratory study of winglets. The chord and twist distributions employed in this section were crafted by the author since the MexNext blade with a winglet modification recorded poor power measurements, with $C_p$ being between 0.10 and 0.20. Also included in the figure is the PrandtlParticleSteadyModel wake, which comprises vortex filaments and a far-wake of vortex particles. The transition of vortex filaments to vortex particles occurs after two helical revolutions of vortex filaments. Truncation of vortex particles is at six helical revolutions, making a total of eight helical revolutions of PF elements.

Table 5.4: Effect of winglet toe angle

<table>
<thead>
<tr>
<th>Winglet Toe Toe (deg)</th>
<th>$C_p$</th>
<th>$C_t$</th>
<th>$C_p/C_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-6</td>
<td>0.458</td>
<td>0.703</td>
<td>0.651</td>
</tr>
<tr>
<td>-3</td>
<td>0.477</td>
<td>0.714</td>
<td>0.667</td>
</tr>
<tr>
<td>0</td>
<td>0.478</td>
<td>0.715</td>
<td>0.668</td>
</tr>
<tr>
<td>3</td>
<td>0.475</td>
<td>0.717</td>
<td>0.663</td>
</tr>
<tr>
<td>6</td>
<td>0.467</td>
<td>0.719</td>
<td>0.650</td>
</tr>
</tbody>
</table>

Table 5.4 presents the summary performance values of the wingleted HAWT rotor for winglet toe angles of -6°, 3°, 0°, 3°, and 6°. The values of $C_t$ indicate that thrust is
reduced by negative winglet toe angles and increased by positive winglet toe angles. Since negative toe angle expands the stream-tube radially, it reduces the flow that passes through the actuator disc. The opposite situation occurs for positive toe angles, which increase flow through the disc. Despite the expected performance gains from winglets with positive toe angles, Table 5.4 indicates that both positive and negative toe angles have an adverse effect on $C_p$ and $C_p/C_t$.

![Graph](image1.png)

(a) Angles-of-attack  

![Graph](image2.png)  

(b) Reynolds number

Figure 5.8: Local flow on blades with winglets

Figures 5.8a and 5.8b examine angles-of-attack and Reynolds numbers along the wingleted blade respectively. Sudden changes occur at $\mu = 0.9$ because that is where the wing transitions into the winglet. The positive inward-toed winglets draw flow inward. Positive toe leads to higher positive angles-of-attack on the winglet. Induced inward flow causes higher angles-of-attack and Reynolds number on the outer portion of the blade near the winglet as well. The outer portion is defined roughly as $\mu$ from 0.8 to 0.9. The opposite assessment holds for negative winglet toe and outwardly induced flow.

Figures 5.9a and 5.9b present the changes in axial and rotational forces brought about by winglet-induced changes in angles-of-attack and Reynolds number. The figures reveal that the winglet itself does not contribute significantly to forces, suggesting that the wing does not need to be made substantially stronger to hold the winglet. The joint of the blade and winglet is expected to require strength and extreme conditions should be considered in winglet design.

The figures show that inward toe angles increase axial force on the outer portion of the blade, causing increased thrust and BRBM. Figure 5.9b reveals that inward
Figure 5.9: Wingleted rotor loadings

toe increases rotational force in the outer portion of the blade as well. However, the increase in rotational force on the blade is almost perfectly offset by an increase in lift-induced drag on the winglet, as seen by the negative rotational force on the winglet portion, defined as $\mu$ being in the region of 0.9 to 1.0. Due to lift-induced drag on the winglet, toeing at any angle reduces rotor power output. For this particular winglet design with 90° upwind orientation, there appears to be no benefit to having winglets. Future work is recommended to explore raked and blended winglet designs in combination with optimized twist distributions.

5.2 Periodic Flow Simulations

Although a detailed analysis of yawed flows and their periodic simulations is beyond the scope of this thesis, a periodic simulation of a standard Danish rotor operating in steady yawed inflow wind conditions is presented in this section. The purpose is to briefly introduce the topic and suggest the direction for future research. In this example, the velocity of the free stream in the crosswise direction is one-half of the velocity in the axial direction. Figure 5.10 presents a HAWT rotor in these conditions as modelled with a periodic vortex filament lifting line and wake model. This particular simulation result derives from an outdated version of LibAero, which lacks most of the output functionality. Future work is intended to bring periodic simulations to the current software line of LibAero. Additionally, validation of the periodic simulator for yawed and shear winds is planned. An evaluation and validation
of the fully unsteady simulator is recommended for future work as well.

Figure 5.10: Snapshot of a steady yaw rotor wake
Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, a C++-based simulation framework for fixed-wing and rotor blade aerodynamics was developed based on potential flow (PF) theory with its major focus on the subset of vortex methods. The simulation framework was developed as a combination of several recent technological advances. The fast multipole method (FMM) produces rapid solutions to N-body dynamics problems, which is the problem category of PF element-based approaches to aerodynamics. Viscous core modelling provides a way to avoid numerical singularities and instabilities in induced flow calculations. Core modelling not only provides robustness, it also provides physically realistic modelling of vorticity transport and provides a zero-equation turbulence solution to the unsteady Reynolds-averaged Navier-Stokes (URANS) equations. Additional techniques such as relaxation iteration and symmetry modelling were employed to reduce the computational expense.

The aim of this line of work is to create a general-purpose aerodynamics solver with applications to the multidisciplinary design optimization (MDO) of wind turbines. The ultimate objective is to use MDO techniques to produce designs of conventional wind turbines as well as design families of unconventional devices such as swept rotors, winglets, vertical-axis wind turbines (VAWTs) and shrouded tidal turbines. Simpler methods such as blade element momentum (BEM) methods are not suitable for complex devices, and computational fluid dynamics (CFD) is computationally expensive. In order to achieve these goals, the simulation tool is required to be computationally speedy and versatile with respect to geometric configurations.
Chapter 4 demonstrated the speed benefits of FMM and relaxation iteration. The FMM approach saves 50% to 75% of computational run time with very little affect on accuracy, having less than 1% error in $C_p$ and $C_t$ values. More aggressive approximations save 88% to 94% of the run time, but are less accurate and less stable. They generally produce errors in the 1-2% range and 2-4% range respectively. A relaxation parameter in the 0.8 to 0.95 range reduced the number of iterations for convergence from roughly twenty to as low as five, saving 75% in computational run times.

Symmetry modelling applies to steady-state, tower-free simulations only. In those cases it models only one blade, thus approximately two-thirds of the run time is saved for a three-bladed turbine simulation. When coupled with FMM and relaxation iteration, symmetry modelling does not offer the full two-thirds time savings because FMM takes up some of the slack. Altogether, FMM, relaxation iteration, and symmetry modelling reduce computational run times to less than 5% of the full simulation values. This level of computational performance enhancement makes PF simulation a competitive alternative to BEM methods. Although LibAero is still slower than typical implementations of BEM methods, its versatility provides sufficient value to justify the choice of LibAero in many circumstances.

Chapter 5 provides a limited demonstration of the geometric versatility of LibAero. The chapter presents LibAero simulation results and comparisons for a three-bladed coned horizontal-axis wind turbine (HAWT) rotor, a swept three-bladed HAWT rotor, and a two-bladed HAWT rotor with 90° upwind winglets. The three simulations are tower-free HAWT rotors in steady wind conditions. The coned rotor simulation results indicate that coning does not offer substantial enhancement of aerodynamic performance, though it may offer control and protection possibilities.

The swept rotor results indicate that swept blades can lead to increased aerodynamic performance, particularly in the case of backward swept blades. It is further suggested that straight blades with a swept tip region should have similar aerodynamic benefits without the structural challenges of a fully bowed blade. This is already seen in some commercial products and experimental designs such as SkyStream$^{TM}$ wind turbines and the Sandia STAR research wind turbine.

The wingleted rotor analysis found that winglets oriented to draw flow into the turbine disc offer performance gains on the outboard portion of the blade, while the aerodynamic loss due to lift-induced drag on the winglet itself gives rise to a net loss. In other words, the 90° winglet design that was arbitrarily chosen for this analysis
did not offer any improvements, although other winglet designs may have potential gains.

The methodology portions of this thesis offered topics for others with interests in understanding and developing PF codes. Chapter 2 presented the URANS derivation in Eulerian form and shows how it develops into a PF element-based Lagrangian method. Derivations of the Biot-Savart equations for each core model, derivations of evolution equations such as advection, diffusion, and vortex strength deformation models, as well as a presentation of the Barnes-Hut tree-code (BHTC) and FMM equations were given. Chapter 3 presented how object-oriented programming (OOP) software design patterns were implemented to reach these final objectives while employing a straightforward software architecture. The major discovery during the progression of this work was that separating numerical grid element data from algorithms was crucial to developing a modular, easily comprehensible code base. The methodology of handling elements with shared endpoints and corner points posed a development challenge as well. This was eventually solved by maintaining duplicate numerical data instead of utilizing pointers to shared data chunks.

Chapter 4, in addition to providing a numerical performance evaluation, also provided experimental and numerical validations. The experimental validation section compared LibAero wake models to the experimental force data and wind tunnel particle image velocimetry (PIV) data of the MEXICO HAWT rotor experiments. The comparison revealed that the best LibAero predictions had an average error measure of 11.8% in $C_p$ and $C_t$ values, while the best ExcelBEM predictions had an average error of 8.0%. For low and medium flow induction, LibAero and BEM performed similarly well in predicting the forces on the MexNext rotor blade. LibAero models had greater error in the high induction case.

The cause of error during high induction scenarios is under investigation. Truncation length, time step, and blade discretization were eliminated as causes through numerical experimentation. Convergence properties are a possible cause, but are not likely to be the issue since fully converged solutions under various settings agree to a large extent. Most likely, the difference is attributable to momentum correction models that are employed by BEM methods. Correction models such as the Glauert model are semi-empirical and involve statistics on a great number of real-world wind turbine cases. As a result, they can account for physical phenomena that are not present in the PF models employed by LibAero, particularly unsteady turbulence effects that occur at high induction [1]. On-going work is necessary to improve the modelling of
vorticity evolution under turbulent conditions in the *LibAero* framework. Flow field velocities simulated with *LibAero* were compared to the MEXICO experimental PIV velocity measurements. Preliminary graphs show a high degree of superficial agreement. Further work is necessary to determine how much of the statistical variation in the comparison of *LibAero* to the MEXICO experimental PIV data can be explained by experimental error and how much is due to simulation error.

### 6.2 Future Work

Although much has been accomplished in this thesis work, it represents a mere scratch on the surface of a larger body of work to come for the purpose of improving wind turbine designs. The most immediate step towards this aim is to tailor the aerodynamic algorithms to match experimental data, possibly involving turbulence and dynamic effects modelling. A great deal of computational speed enhancement was made using FMM, relaxation iteration, and symmetry models. Further improvements are expected by incorporating multigrid methods, general-purpose computing on graphics processing units, and far-wake approximations.

Multigrid methods start the simulation on a course grid to estimate numerical values, then carry on to finer grids to reach a final solution. This method saves time that would be spent computing inaccurate data on a detailed grid in the early iteration stages. General-purpose computing on graphics processing units (GPGPU) involves creating a parallel algorithm to be piped through the many floating point units of graphics cards. Both FMM and the evolution equations have promising parallel implementations. Finally, a computationally efficient far-wake model that allows the near-wake elements to be truncated earlier would greatly enhance computational speed. A semi-infinite vortex model has been in planning, but there are issues with expressing the near- to far-wake transition equations. There are challenges in deriving analytic expressions for the induced flow of semi-infinite vortex elements as well.

Further steps are needed to validate tower shadow and collision models. Tower shadow modelling typically involves the simple strategy of superimposing a flow element downwind of the tower to represent the velocity deficit due to flow separation [29]. The collision models displace PF elements that protrude solid objects and relocate them strategically to the outside of the object. The periodic simulation is needed for simulation of HAWTs rotors with tower, wind shear, and yaw. Periodic simula-
tion is also necessary for VAWTs simulations under any conditions. The unsteady simulation is necessary for unsteady winds and for multi-rotor devices. Simulation of several devices such as a wind farm is also a possibility, although computational performance scaling does not suggest this to be a primary role for *LibAero*.

Finally, combining *LibAero* models with aeroelastic structural models, control systems, acoustics, and electric systems is a necessary step toward the ultimate aim of designing improved devices. At that stage, MDO techniques can provide the glue that will allow improved designs and whole design families to be discovered in an efficient manner.
Bibliography


