NOTE TO USERS

This reproduction is the best copy available.
RICE UNIVERSITY
Computational Techniques For Aerodynamic Simulations Of Multiple Objects Emphasizing Paratrooper-Aircraft Separation
by
Victor Udoewa
A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE DOCTOR OF PHILOSOPHY
APPROVED, THESIS COMMITTEE:

Enrique Barrera, Chair, Professor, Mechanical Engineering and Material Science

Richard Tapia, Co-chair, Noah Harding Professor, Computational and Applied Mathematics

Yildiz Bayazitoglu, H. S. Cameron Professor, Mechanical Engineering and Material Science

William Symes, Noah Harding Professor, Computational and Applied Mathematics

Houston, Texas
April, 2005
INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI Microform 3168150
Copyright 2005 by ProQuest Information and Learning Company. All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346
Abstract

Computational Techniques for Aerodynamic Simulations between Multiple Objects Emphasizing Paratrooper–Aircraft Separation

by

Victor Udoewa

Our target is to develop computational techniques for studying aerodynamic interactions between multiple objects with emphasis on studying the fluid mechanics and dynamics of an object exiting and separating from an aircraft. The object could be a paratrooper jumping out of a transport aircraft or a package of emergency aid dropped from a cargo plane. These are applications with major practical significance, and what I learn and what I develop can make a major impact on this technology area. In all these cases, the computational challenge is to predict the dynamic behavior and path of the object, so that the separation process is safe and effective. This is a very complex problem because it has an unsteady, three–dimensional nature and requires the solution of complex equations that govern the fluid dynamics of the object and the aircraft together, with their relative positions changing in time.

The gravitational and aerodynamic forces acting on the object determine its dynamics and path. Sometimes those aerodynamic forces are not properly computed due to excessively thick numerical boundary layers (numerical meaning unphysical and unreal). Methods for reducing this thickness are presented here. The aerodynamic forces heavily depend on the unsteady flow field around the aircraft. The
computational tools I am developing are based on the simultaneous solution of the
time–dependent Navier-Stokes equations governing the airflow around the aircraft and
the separating object, as well as the equations governing the motion of that object.
These computational methods include suitable mesh update techniques that are essen-
tial for simulations with my core computational technique – the Deforming–Spatial-
Domain/Stabilized Space–Time (DSD/SST) formulation. In the research I present
here, I focus on developing mesh update methods that help me perform my com-
putations with more numerical accuracy and better computational efficiency. These
methods range from remeshing tactics with reduced distortion, to methods reducing
the error introduced through projection and, finally, even to a mesh moving alter-
native – Fluid Object Interaction Subcomputation Technique (FOIST). In FOIST,
moving object problems are computed with an approximation technique, without the
costs of mesh moving, remeshing, or projection.
Acknowledgments

“For I know the plans I have for you,” declares the Lord, “plans to prosper you and not to harm you, plans to give you hope and a future.” (Jeremiah 29:11)

My paramount debt of gratitude goes to God. Throughout my graduate school tenure, the most important part was the spiritual aspect. Without that in check, I would have left long ago. I am truly blessed. Thank you.

To my AGEP family, I am deeply indebted. You were not only financial support, but an encouraging community, a constructive social support group, and a family. Thank you for all the laughs and memories.

Next, I would like to thank my former advisor, Professor Tayfun Tezduyar, for originally allowing me to work in his lab.

Next, I would like to thank my advisor, Professor Richard Tapia for his saving help and willingness to take me in as one of his own original students. Thank you. Your help has beneficially affected my present and my future.

Thank you to my co-advisors who helped me with everything from facts and figures to code bugs: Dr. Andrew Johnson, Richard Benney, Dr. Marek Behr, and Dr. Keith Stein. Your multi-leveled help was invaluable.

My team, my lab, my crew—you helped me get through every day at Rice. And each day brought us closer. Thank you, Weiqing Yee, Vinod Kumar, Mehmet Unger, Medhat El-Nahas, Tomas Soltys, Ryan Keedy, Rachel Melton, Lea Haubelt, Dhruv Arora, Feby Abraham, Sunil Sathe, Masayoshi Senga, Cengizhan Bilgın, Omar Albaity, Adel Alshayji, Yiqun Hwang, Randy, and Darby Vicker.
To my friends outside of the lab, on-campus at Rice and off, my community and city, my Calvary church, my Houston Akwa Ibom community, and around the world—I can never say enough. You are too numerous for me to attempt to name, for I will doubtlessly forget some. Still, thank you, wholeheartedly. You were constant encouragement and consummate motivation to me. Everything you think I gave to you, I received from you. As they say, it takes a whole village to raise a child, and you showed me I’m still growing.

Last, and most important after God, I thank you, Mom, Dad, for everything I’ve said and things I’ve never said. I’m here because of you and your prayers. Jane and Patrick, thank you. They say family is a crutch, but you guys were my legs and air.
# Contents

Abstract 1  

Acknowledgments iv  

List of Tables x  

List of Figures xiv  

1 Introduction 1  
  1.1 Overview .................................................. 5  
  1.2 Contributions ............................................. 6  

2 Governing Equations 9  
  2.1 Fluid Dynamics ............................................. 9  
    2.1.1 Navier–Stokes Equations of Incompressible Flows .......... 9  
    2.1.2 Stress and Strain Definitions ............................. 10  
    2.1.3 Boundary and Initial Conditions ......................... 10  
    2.1.4 Turbulence Model ...................................... 11  
  2.2 Object–Related Equations ................................. 12  
    2.2.1 Forces Acting on Objects ................................ 12  
    2.2.2 Particle Dynamics ..................................... 14  

3 Finite Element Formulations 16  
  3.1 Semi–Discrete Formulation ................................. 16
3.1.1 Interpretation of the Terms .............................................................. 17
3.1.2 Time Derivatives ................................................................................. 18
3.2 Space-Time Formulation ........................................................................ 19
  3.2.1 Interpretation of the Terms ................................................................. 21
  3.2.2 Space–Time Elements ........................................................................ 22
3.3 Nonlinearity .............................................................................................. 23
3.4 Stabilization .............................................................................................. 24
3.5 Implementing Particle Dynamics ............................................................... 24

4 Mesh Generation and Update ..................................................................... 27
  4.1 Mesh Generation ..................................................................................... 28
    4.1.1 Previous Work .................................................................................. 28
    4.1.2 New Contributions ........................................................................... 29
  4.2 Mesh Moving Overview: Previous Work ............................................... 29
    4.2.1 One Remesh Per Time Step ............................................................... 32
    4.2.2 Specialized Design .......................................................................... 33
    4.2.3 Automatic Mesh Moving: Linearly Elastic Model and Formu-
          lation ................................................................................................ 34
    4.2.4 Automatic Mesh Moving With Non-moving Zones ......................... 36
  4.3 3–D Mesh Moving Strategies .................................................................... 37
    4.3.1 Jacobian–based Stiffening: Previous Work ....................................... 39
    4.3.2 Jacobian–based Stiffening: New Work ............................................. 40
    4.3.3 Variable Stiffening: Previous Work ................................................. 41
    4.3.4 Variable Stiffening: New Work ....................................................... 42
    4.3.5 Strain–Based Stiffening: Previous Work .......................................... 46
    4.3.6 Strain–Based Stiffening: New Work ................................................. 46
    4.3.7 Material Property Stiffening: Previous Work ................................... 47
    4.3.8 Material Property Stiffening: New Work ......................................... 47
    4.3.9 Aspect Ratio–Based Stiffening: New Contribution ............................. 47
  4.4 Traditional Remeshing .......................................................................... 48
4.5 Projection ................................................................. 49
  4.5.1 Conventional Projection ........................................ 50
  4.5.2 Compressibly Adjusted Interpolation: A New Tactic .... 53
  4.5.3 A New Projection Method: Divergence–Free Projection ... 55

5 Fluid–Object Interactions Subcomputation Technique 71
  5.1 New FOIST Method .................................................. 72
  5.2 FOIST Issues .......................................................... 73
    5.2.1 FOIST Trajectory Starting Point ............................ 73
    5.2.2 Outflow Boundary Conditions ............................... 73
    5.2.3 Conflicting Boundary Conditions ........................... 74
  5.3 Applications .......................................................... 75
    5.3.1 Ball in Uniform Flow ......................................... 76
    5.3.2 Ball in Shear Flow ............................................ 76

6 Boundary Layer Resolution 81
  6.1 Boundary Layer Elements: Overview of Current Problems ... 82
    6.1.1 Overview of Current Problems .............................. 82
    6.1.2 Boundary Layers: New Contributions ...................... 84
  6.2 Slip Boundary Condition ......................................... 94
    6.2.1 New Contribution: Implementation of Slip Boundary Condition 94
    6.2.2 Slip Boundary Applications .................................. 100

7 Paratrooper Separating from an AirForce Cargo Aircraft 103
  7.1 Fluid Model ......................................................... 105
  7.2 FOI Setup ............................................................ 108
  7.3 FOI Simulation ....................................................... 109
    7.3.1 Slip Simulation ............................................... 111
    7.3.2 FOIST Simulation ............................................. 112
8 Cargo Payload Separating from an AirForce Cargo Aircraft 124
  8.1 Fluid Model ........................................ 125
  8.2 FOI Setup ........................................... 128
  8.3 FOI Simulation ....................................... 129
    8.3.1 Slip Simulation ................................. 131
    8.3.2 FOIST Simulation ................................. 132

9 Conclusions ............................................ 139

A Appendix .............................................. 144

Bibliography ............................................ 149
List of Tables

4.1 First Paratrooper Simulation: Time Steps per Mesh. 65
4.2 First Paratrooper Simulation: Time Steps per Mesh—Continued. 66
4.3 Second Paratrooper Simulation: Time Steps per Mesh. 67
4.4 Second Paratrooper Simulation Time Steps Per Mesh—Continued. 68
4.5 Second Paratrooper Simulation Time Steps Per Mesh—Continued. 69
4.6 Paratrooper Simulation: Variable Stiffening. 69
4.7 Paratrooper Simulation: Strain–Based Stiffening. 69
4.8 Paratrooper Simulation: Material Stiffening. 70
4.9 Paratrooper Simulation: Aspect Ratio–Based Stiffening. 70
## List of Figures

3.1 Space–Time Concept for a 2–D Spatial Domain. .......................... 20  
4.1 Space–time Slabs: Remeshing Every Time Step. ......................... 32  
4.2 Space–time Slabs: Maintained Connectivity. .......................... 33  
4.3 Ball in Shear Flow: 3D Surface Meshes. .............................. 35  
4.4 Ball in Shear Flow: 3D Surface Mesh. ................................. 38  
4.5 Stiffness Coefficient Dependence on Element Volume. ............... 42  
4.6 Stiffness Coefficient Dependence on Element Volume. ............... 43  
4.7 Stiffness Coefficient Dependence on Element Volume. ............... 44  
4.8 Stiffness Coefficient Dependence on Element Volume. ............... 45  
4.9 Freestream Component of Force Felt by Paratrooper: Clip Once Adding Extra Non-Linear Iteration. ........................................ 55  
4.10 Freestream Component of Force Felt by Paratrooper: One Clip Moving Immediately to Next Time Step. .............................. 56  
4.11 Freestream Component of Force Felt by Paratrooper: Two Clips. .... 57  
4.12 Freestream Component of Force Felt by Paratrooper: Four Clips. .. 57  
4.13 Freestream Component of Force Felt By Paratrooper: 0/1/2/4 Clips. 58  
4.14 Nodal Block in Matrix for Divergence–Free Projection. ............... 61  
4.15 Remeshing using a Divergence–Free Projection Solver. ............... 63  
4.16 Non-linear Time Steps Before and After Remeshing for a Paratrooper Falling from a Cargo Aircraft. .............................. 64  
5.1 3D Surface Meshes of Flow Around a Ball. ............................ 77
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>3D Surface Meshes of Flow Around a Ball.</td>
<td>78</td>
</tr>
<tr>
<td>5.3</td>
<td>Trajectory of Ball in Uniform Flow: FOIST vs Mesh Moving.</td>
<td>79</td>
</tr>
<tr>
<td>5.4</td>
<td>Trajectory of Ball in Shear Flow: FreeStream vs Gravity Direction.</td>
<td>80</td>
</tr>
<tr>
<td>6.1</td>
<td>3D Surface Mesh of AirForce Cargo Aircraft.</td>
<td>83</td>
</tr>
<tr>
<td>6.2</td>
<td>3D Surface Mesh of Aircraft and Domain Box: Front View.</td>
<td>84</td>
</tr>
<tr>
<td>6.3</td>
<td>3D Surface Mesh of Aircraft and Domain Box: Side View.</td>
<td>85</td>
</tr>
<tr>
<td>6.4</td>
<td>3D Surface Mesh of Highlighted Aircraft and Domain Box: Side View.</td>
<td>86</td>
</tr>
<tr>
<td>6.5</td>
<td>Color Plot of Velocity in Boundary Layer.</td>
<td>87</td>
</tr>
<tr>
<td>6.6</td>
<td>Color Velocity Plot in Boundary Layer with Double Refinement on the</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td>Aircraft Surface.</td>
<td></td>
</tr>
<tr>
<td>6.7</td>
<td>Boundary Layer Around Fuselage of Cargo Aircraft.</td>
<td>89</td>
</tr>
<tr>
<td>6.8</td>
<td>Boundary Layer Velocity Surface Around Fuselage of Cargo Aircraft:</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>Side View.</td>
<td></td>
</tr>
<tr>
<td>6.9</td>
<td>Boundary Layer Velocity Surface Around Fuselage of Cargo Aircraft.</td>
<td>92</td>
</tr>
<tr>
<td>6.10</td>
<td>Boundary Layer Velocity Surface Around Fuselage of Cargo Aircraft:</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td>Side View.</td>
<td></td>
</tr>
<tr>
<td>6.11</td>
<td>Velocity in Free-Stream Direction About Ball with Slip Boundary</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>Conditions.</td>
<td></td>
</tr>
<tr>
<td>7.1</td>
<td>Paratroopers Preparing to Exit from Cargo Aircraft.</td>
<td>103</td>
</tr>
<tr>
<td>7.2</td>
<td>Cargo Aircraft in Formation with Paratroopers Exiting.</td>
<td>104</td>
</tr>
<tr>
<td>7.3</td>
<td>Original Cargo Aircraft.</td>
<td>106</td>
</tr>
<tr>
<td>7.4</td>
<td>Updated Cargo Aircraft Surface Mesh.</td>
<td>107</td>
</tr>
<tr>
<td>7.5</td>
<td>Updated Cargo Aircraft Surface Mesh—Posterior View.</td>
<td>107</td>
</tr>
<tr>
<td>7.6</td>
<td>Original Paratrooper Model.</td>
<td>108</td>
</tr>
<tr>
<td>7.7</td>
<td>Updated Paratrooper Model.</td>
<td>113</td>
</tr>
<tr>
<td>7.8</td>
<td>Updated Paratrooper Model with Parachute Pack.</td>
<td>114</td>
</tr>
<tr>
<td>7.9</td>
<td>Trajectory Snapshots of Original Paratrooper Model.</td>
<td>115</td>
</tr>
<tr>
<td>7.10</td>
<td>Old Trajectory of New Paratrooper Model.</td>
<td>116</td>
</tr>
<tr>
<td>Section Description</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>7.11. Pressure Distribution on Cargo Aircraft</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>7.12. Streamribbons about Cargo Aircraft Colored with Pressure.</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td>7.13. Pressure Distribution on Paratrooper with Streamlines.</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td>7.15. Paratrooper Trajectory Comparison: Real vs Numerical vs Analytical</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>7.16. Paratrooper Trajectory with Slip Conditions on Aircraft.</td>
<td>121</td>
<td></td>
</tr>
<tr>
<td>7.17. Paratrooper Trajectory Comparison: Real vs Numerical vs Analytical vs Slip</td>
<td>122</td>
<td></td>
</tr>
<tr>
<td>7.18. Model of FOIST Subcomputational Domain Containing Paratrooper</td>
<td>122</td>
<td></td>
</tr>
<tr>
<td>7.19. Paratrooper Trajectory Comparison: Real vs Numerical vs Analytical vs Slip vs FOIST</td>
<td>123</td>
<td></td>
</tr>
<tr>
<td>8.1. Payloads Falling from Bay of Cargo Aircraft</td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>8.2. Payloads Falling from C-130 Aircraft.</td>
<td>126</td>
<td></td>
</tr>
<tr>
<td>8.3. New Cargo Aircraft Model with Cargo Bay.</td>
<td>126</td>
<td></td>
</tr>
<tr>
<td>8.4. Cargo Aircraft with Cargo Bay–Side View.</td>
<td>126</td>
<td></td>
</tr>
<tr>
<td>8.5. Cargo Payload Prototype.</td>
<td>127</td>
<td></td>
</tr>
<tr>
<td>8.6. Cargo Payload Model.</td>
<td>127</td>
<td></td>
</tr>
<tr>
<td>8.7. Cargo Payload Model–Second View.</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>8.8. Pressure Distribution on Cargo Payload and Aircraft.</td>
<td>129</td>
<td></td>
</tr>
<tr>
<td>8.9. Streamribbons About Cargo Payload.</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td>8.10. Streamribbons About Cargo Payload–Side View.</td>
<td>131</td>
<td></td>
</tr>
<tr>
<td>8.11. Surface Mesh of Cargo in Pre-Tipping Stage.</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>8.12. Cargo Just Before Tipping Over Cargo Bay Door.</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>8.13. Trajectory Snapshots of Payload Drop Simulation.</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>8.15. Payload Trajectory with Slip Conditions on Aircraft.</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>8.16. Last Snapshot of Payload Trajectory with Slip Conditions on Aircraft</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>8.17. Cargo Payload Trajectory Comparison: Numerical vs Analytical vs Slip</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>8.18. Model of FOIST Subcomputational Domain Containing Payload.</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>8.19. Side View of Colored Payload in Subcomputational Domain for FOIST</td>
<td>137</td>
<td></td>
</tr>
</tbody>
</table>
8.20 Cargo Payload Trajectory Comparison: Numerical vs Analytical vs Slip vs FOIST. .................................................. 138

9.1 Paratroopers Exiting Cargo Aircraft. ........................................ 139

A.1 Trajectory of Ball-Shear Flow: FreeStream vs Transverse Directions. . 144
A.2 Trajectory of Ball in Shear Flow: Gravity-Direction vs Transverse Direction. 145
A.3 Red - Freestream Value vs Blue < 100% Freestream Value. ............. 145
A.4 Dichotomous Velocity Picture: Red ≥ 90% Freestream Value, Blue < 90%
Freestream Value. ........................................................... 146
A.5 Dichotomous Velocity Picture: Red ≥ 90% Freestream Value, Blue < 90%
Freestream Value. ........................................................... 146
A.6 Dichotomous Velocity Picture: Red ≥ 85% Freestream Value, Blue < 85%
Freestream Value. ........................................................... 147
A.7 Dichotomous Velocity Picture: Red ≥ 80% Freestream Value, Blue < 80%
Freestream Value. ........................................................... 147
A.8 Dichotomous Velocity Picture: Red ≥ 75% Freestream Value, Blue < 75%
Freestream Value. ........................................................... 148
A.9 Effect of Inertia Scalar (Left) vs Inertia Matrix (Right). ................. 148
Chapter 1

Introduction

Paratroopers jump from a variety of different aircrafts weighted with different supplies, depending on the mission, and experiencing an array of weather conditions. These meteorological conditions, as well as the shape of the aircraft, the size of the aircraft, and turbulence caused by other aircraft in the vicinity can affect the free fall path of the paratrooper.

The Army has experienced a myriad of problems associated with this situation. Two examples are crossover and wake crossing. With a specific AirForce aircraft, the Army has had difficulties whenever paratroopers jump out of both the left and the right doors simultaneously; their paths tend to cross-crossover. This is, of course, potentially injurious. In wake crossing, with multi-aircraft jumps, as paratroopers jump out of AirForce aircraft, they must fall through the wake of the preceding aircraft. If that wake has not experienced sufficient dissipation, fatalities could occur which has happened due to the difficulty in pulling parachute chords in that turbulent wake.

Obvious solutions include alternating left door and right door paratrooper jumps and flying all aircraft very far apart. But the Army has highly time-sensitive missions during which they need to exit all paratroopers in as short a time as possible using as little air space as possible. Therefore, they prefer to have paratroopers jumping simultaneously and want to know the safest closest distance these aircraft can be
flown to avoid wake-crossing casualties.

But paratrooper–aircraft separation problems are not the only ones with which the Army is concerned. Cargo drops also present a problem. Many times, the Army may want to make a supply drop to an operative in another country. Or they want to drop off a shipment to a group of soldiers. In those cases, the Army wants to be able to pinpoint the location of a drop within a certain radii, so that it can be easily and quickly found. For that reason cargo drops are also of interest. Cargo drops, as with paratrooper jumps, come in a wide variety of forms. Some use an exit parachute to extract it out. Others slide out on frictionless tracks with the help of gravity, only. The ability to predict the trajectory of a certain payload and its target can highly expedite many operations.

Wind tunnel experiments can even be more inflexible when dealing with paratrooper jumps. How do you model a falling paratrooper in a wind tunnel? And how do you do it realistically? Moreover, experimental data from actual jumps is also hard to obtain. The limitations of experimental research in this area are clear. First, the cost can be prohibitive. These include labor, soldier hours, pilot hours, fuel, aircraft rental, airfield rental, etc. Secondly, the time can be restrictive. One test jump can be planned for over an entire year! And the amount of personnel needed creates inconvenience. Numerical modeling attacks all of those problems. Wind tunnel experiments can even be more inflexible, especially when dealing with paratrooper jumps. It is much cheaper, expends less time for the same experiment, and is more convenient, requiring less people (feasibly one). And as computer science technology advances, amazing leaps in computer power and memory continue to make larger and larger 3-D simulations easier to tackle, calculating, in minutes, simulations that previously took hours to compute.

Here, I explore simulation and modeling techniques for Fluid–Object Interactions, FOI, aerodynamic interactions between multiple objects. All objects will be treated as rigid bodies. The specific application of the paratrooper–aircraft separation will be emphasized. The FOI strategy consists of three components: the fluid dynamics (FD) simulation, the Newtonian particle–force calculations, and the mesh moving
calculations.

This is a very complex problem because it has an unsteady, three-dimensional nature. Moreover, the complexity increases because the problem requires the solution of large systems of nonlinear equations governing the fluid dynamics of the object and the aircraft, together, and with their relative positions changing in time.

The gravitational and aerodynamic forces acting on the object determine its dynamic behavior and path. The aerodynamic forces greatly depend on the unsteady flow field around the aircraft. The computational tools developed, here, are based upon the simultaneous solution of the 3-D time-dependent Navier–Stokes equations governing the incompressible airflow around the aircraft and the separating object, as well as the equations governing the motion of that object. These computational methods include suitable mesh update techniques to be used in conjunction with my core computational technique, the Deforming–Spatial–Domain/Stabilized Space–Time (DSD/SST) formulation introduced by Tezduyar [1–3]. Previously, techniques such as Arbitrary Lagrangian–Eulerian formulation were used for moving problems with both finite difference modeling, finite volume modeling [4–7] and finite element modeling [8–11]. Because the relative positions of the aircraft and the separating object are changing in time, the Navier–Stokes equations governing the motion of the surrounding air need to be solved over a computational domain that changes in shape and time. The DSD/SST formulation is written over the corresponding space–time domain of the problem, and can, therefore, automatically handle the changes in the spatial domain. Recently, this method has been tested on many problems, including those involving 3-D domains, high Mach numbers and high Reynolds number flows, and moving boundaries and interfaces [12–14]. In this thesis, the formulation is applied to mesh moving problems with fluid object interaction applications.

Effective mesh update methods, to be used in conjunction with the DSD/SST formulation, are essential for accurate computation of this complex problem. I focus on new mesh update methods that will help me perform my computations with more numerical accuracy and better computational efficiency. In moving the mesh, I can choose to move the mesh in any manner. The choice is arbitrary. Here I employ the
equations of linear elasticity in solving the mesh’s movement. This solid mechanics
component solves these equations of linear deformation over the prescribed moving
part of the domains for the object separation problems.

With mesh moving problems, comes the difficulty of remeshing and projection.
As a mesh moving fluid dynamics simulation runs, the mesh continuously becomes
more distorted. According to a suitably set distortion limit, the simulations stop
when that limit is reached. A new 3-D mesh is generated for the same space, and
the solution from the previous mesh is projected onto the new mesh. This usually
introduces errors evidenced by jumps in the pressure resulting in jumps in particle
trajectory, displacements, and velocities at the temporal points of a remesh in the
simulation. The errors come from a violation of the incompressibility constraint.
After a remesh, as the simulation continues, the computation gradually returns to
previous values before the jump. New tactics to reduce the remeshing jumps and new
methods to account for the incompressibility constraint and eliminate the jumps all
together will be presented. Of these new contributions, one of the latter methods
utilizes the Conjugate Gradient Method for the Normal Equations (CGNE) on an
overdetermined system of linear equations using an LQ preconditioner [15]. The
system is overdetermined due to the constraint of incompressibility on the projection
equation system.

The mesh moving calculations, remeshing, projection errors, and resulting jumps
and errors in the solution can be costly, especially as they compound each other. For
this reason, many have looked to developing alternatives to mesh moving. As a new
contribution, the Fluid–Object Interactions Subcomputation Technique (FOIST),
does exactly this [16]. It is an intermediate approximation between the treatment
of the particles as point masses and a full fluid dynamics mesh moving computation.
With the assumption that the particle or particles have negligible effects on the com-
putational flow, the computational flow is calculated once. Subsequently, information
is continuously extracted from the main computational flow and used to calculate the
flow around the specific particle in a subcomputational domain. Results from the
fixed subcomputation are used to find the new location of the particle in the main
computation which becomes the new extraction site.

In Fluid–Object Interactions involving separation, the boundary layer plays a large role in determining the path of the paratrooper. To perform an accurate simulation, the size of the numerical boundary layer must match that of the actual boundary layer of the aircraft at the specified Reynolds number. Otherwise, improper forces inadequately propel the particles off course from the experimental trajectory. Increasing the refinement becomes only a short-term solution as the problems grow larger and larger. Adaptive meshing has become a crucial topic because of this. Here I explore a newly contributed solution to this problem through the creation of tetrahedral boundary layers for arbitrary surfaces. This new capability is greatly needed for many of my problems, and its effectiveness will be tested. But most assuredly, I have also employed the new capability of slip boundary conditions over arbitrary surfaces. This is an excellent approximation in flows with extremely high Reynolds numbers and severely thin boundary layers.

1.1 Overview

I commence with the governing equations for fluid flow. The fluid flow is governed by the Navier–Stokes equations for incompressible flow. Chapter 2 is a presentation of those equations with the constitutive relations and the boundary and initial conditions. Instead of directly resolving the turbulent flow features present at the Reynolds numbers of the problems presented in this thesis, I account for the turbulence affects using a zero-equation Smagorinsky turbulence model [17]. Lastly, I present other equations related to fluid flow and particles for my specific applications.

The finite element formulations for the FD governing equations are explicated in Chapter 3. In this chapter, I first present the semi-discrete finite element formulation. Then I outline the DSD/SST method for the fluid dynamics and my solution strategy. Finally, the implementation of these finite element formulations is discussed along with related equations.

Mesh generation and update is presented in Chapter 4. Various challenges as-
associated with the handling of deforming domains in FOI are identified. Automatic mesh moving strategies for handling deforming boundaries are presented. Various new methods to minimize distortion, reduce and eliminate projection errors, and rectify remeshing jumps are discussed.

In Chapter 5, the FOIST methodology is explained. Other methods are briefly explained, and I demonstrate how this is an intermediate step with savings from both extremes. I outline the procedure and strategy, highlighting a few examples. Finally, I discuss variants of this new technique.

Boundary layer creation is discussed in Chapter 6. The problem of boundary layer resolution and subsequent minimization is elucidated as motivation. Alternatively, the newly applied approximation of slip boundary conditions for high Reynolds number flows is presented. These slip conditions are now used on arbitrary surfaces. A small test example is given for validation. And then its effect on the boundary layer of my AirForce aircraft is observed.

In Chapters 7 and 8, the new strategies described in Chapters 4–6 are applied to the FOI problems of a paratrooper and a cargo payload separating from an AirForce cargo aircraft–new applications. This payload will be extracted by gravity only; no extraction chute will be used. Comparisons are made with experimentally measured data in order to perform validations of the mesh update methods, FOIST, and the slip and boundary layer codes. Unless otherwise stated, computations were carried out on a CrayT3E parallel supercomputer. A finalizing overview of contributions are made in Chapter 9. Conclusions and implications are drawn from the research. This is followed by proposed areas for future, further, and continuing research.

1.2 Contributions

At the beginning, it is important to demonstrate the areas of new research in this thesis to illustrate my contributions. It also uplifts my originality and creativity. Therefore, my contributions are in four main areas.

- Mesh Generation and Update
- I developed and tested new mesh stiffening tactics including the use of various variable stiffening curves.
- I developed and tested a new mesh stiffening tactic—aspect-ratio—based stiffening that works better than all others tested including variable, Jacobian-based, material-based, and strain-based stiffeners.
- I developed a new method of clipping, different than the original archetype, that conclusively reduces pressure spikes due to remeshing and projection.
- I developed a new projection solver that utilizes the divergence-free constraint to eliminate pressure spikes due to remeshing and projection.

- **FOIST**

  - I developed this new technique which is a hybrid between a full mesh moving simulation and a point–force simulation.
  - I succeeded in producing results that approximate a general trajectory of moving objects in separation problems.

- **Boundary Layer Resolution**

  - I wrote a boundary layer generation program that creates a 3D tetrahedral boundary layer on arbitrary surfaces.
  - I implemented a slip boundary condition in my 3D Navier–Stokes flow solver using a new angle–based method for calculation of point–normals in 3D.

- **Aircraft–Paratrooper and Aircraft–Payload Application**

  - I obtained a reasonable and viable trajectory of a paratrooper jumping from an AirForce cargo aircraft.
  - I applied the aforementioned boundary layer resolution, specifically a 3D slip boundary condition, to obtain an even more realistic trajectory.
- I applied the earlier mesh stiffening tactics and successfully reduced remeshing for both separation applications.

- I applied the new projection method and eliminated pressure spikes due to numerical integration across disparate meshes.

- I applied FOIST to these problems to predict a general trajectory shape for these and other such separation problems.

It is important to note that contributions can be in areas of new method development or in areas of implementation or application. In the area of CFD, implementation and application are sizeable and considerable challenges that constitute new work and research, as well. All real pictures were obtained by permission of the US Army and US AirForce either directly or through their web sites. Some numerical pictures were recreated or created from what data should and would look like if it were accessible.
Chapter 2

Governing Equations

In this chapter, the equations governing the fluid dynamics are presented. The fluid is assumed to be incompressible and will be modeled with the Navier–Stokes equations of incompressible flow. The constitutive relationships, boundary and initial conditions, and the particle–related equations are subsequently included.

2.1 Fluid Dynamics

In this thesis, the dynamics consist of those related to the fluid and those of the particles. Each set of dynamics equations affects the others. For example, in air, the aerodynamics will determine the forces a jumping paratrooper feels acting upon him or her. This in turn affects the path of the paratrooper. I start by examining the fluid dynamics.

2.1.1 Navier–Stokes Equations of Incompressible Flows

Let $\Omega_t \subset \mathbb{R}^{n_{sd}}$ be the spatial fluid mechanics domain and $(0, T)$ be the temporal domain, where $n_{sd}$ is the number of space dimensions, letting $\Gamma_t$ denote the boundary of the spatial domain $\Omega_t$. The subscript “t” evinces the time–dependence of the spatial domain and its boundary. The spatial and temporal coordinates are denoted
by \( x = (x, y, z) \) and \( t \in (0, T) \). The Navier–Stokes equations of incompressible flow can be written as:

\[
\rho \left( \frac{\partial u}{\partial t} + u \cdot \nabla u + f \right) - \nabla \cdot \sigma = 0 \quad \text{on } \Omega_t \quad \forall t \in (0, T),
\]

\[
\nabla \cdot u = 0 \quad \text{on } \Omega_t \quad \forall t \in (0, T). \tag{2.1} \tag{2.2}
\]

where \( \rho \) is the constant fluid density, \( f \) is an external force term such as gravity, and \( \sigma \) is the stress tensor. Equation 2.1 ensures the conservation of momentum for the fluid system; Equation 2.2, the conservation of mass. The flow variables that are being solved for are the velocity, \( u(x, t) \); and pressure, \( p(x, t) \), which is embedded in the stress tensor defined in the next section.

### 2.1.2 Stress and Strain Definitions

I consider Newtonian fluids, and thus assume a linear relationship between the fluid stress and strain tensors. The stress tensor is written as the sum of its isotropic and deviatoric parts:

\[
\sigma(u, p) = -pI + 2\mu\varepsilon(u), \tag{2.3}
\]

where \( \mu \) is the dynamic viscosity coefficient, \( I \) is the identity tensor, \( p \) is the degree of freedom corresponding to pressure, and \( \varepsilon(u) \) is the strain rate tensor:

\[
\varepsilon(u) = \frac{1}{2} \left( \nabla u + (\nabla u)^T \right). \tag{2.4}
\]

### 2.1.3 Boundary and Initial Conditions

To appropriately represent the fluid dynamics, proper boundary conditions must be imposed on the outer boundaries of the fluid domain and on any inner surfaces, ensuring an accurate fluid environment. Both the (essential) Dirichlet– and (natural) Neumann–type boundary conditions are accounted for, and are represented as

\[
u = g \quad \text{on } (\Gamma_t)g,
\]

\[
n \cdot \sigma = h \quad \text{on } (\Gamma_t)h, \tag{2.5}
\]
where \((\Gamma_t)_g\) and \((\Gamma_t)_h\) are complementary subsets of the boundary \(\Gamma_t\) related to the Dirichlet and Neumann–type boundary conditions, respectively.

There are three types of flow boundary conditions that are normally imposed.

- A **prescribed** boundary condition is imposed on inflow boundaries and fluid–structure interface boundaries. In this case, a Dirichlet–type boundary condition is imposed for each of the three velocity components. Prescribed velocities for fluid–structure interface boundaries are obtained from the structural dynamics behavior. This boundary type can also be used to specify an incoming velocity profile or to designate a no-slip boundary condition on an object.

- **Slip** boundary conditions are often imposed on the side "walls" of the fluid domain. Here, the normal velocity component is set to zero by imposing a Dirichlet–type boundary condition, and a Neumann–type boundary condition is imposed in the tangential directions. Thus, the flow is allowed to "slip" in the tangential directions, on the outer planes of the fluid domain. This condition is employed for symmetry planes and boundaries or for frictionless walls bounding the fluid domain.

- A **traction–free** boundary condition is imposed on outflow boundaries and free surfaces. In these cases, a Neumann–type boundary condition is imposed in each direction, leaving each velocity component free. The homogeneous Neumann–type boundary condition \((\mathbf{h} = 0)\) is imposed as the exit flow conditions.

In time–dependent problems, an initial condition is required. The initial condition on the velocity is specified as \(\mathbf{u}(\mathbf{x}, 0) = \mathbf{u}_0\) on \(\Omega_0\), where \(\mathbf{u}_0\) is divergence free.

### 2.1.4 Turbulence Model

For the problems presented here, the viscosity, \(\mu\) is modified locally using a Smagorinsky turbulence model [17]. The viscosity is replaced by an "effective" turbulent viscosity \(\mu_t\) as

\[
\mu_t = \mu + (\kappa h_c)^2 \sqrt{2 \mathbf{e}(\mathbf{u}) : \mathbf{e}(\mathbf{u})},
\]

(2.6)
where $\kappa = 0.15$, and $h_e$ is a measure of the element length.

### 2.2 Object–Related Equations

In many fluid flow problems, there are other equations needed and introduced to account for various phenomena related to the flow. In Fluid–Object Interaction problems, the equations of the fluid forces acting on the objects and the object dynamics are required, since the object trajectory and dynamics affect the flow field which in turn influence the object through fluid dynamic forces.

Object–object forces will be ignored. In the applications to be presented, no collisions between particles or objects were allowed. Steps were taking to model the effect of such interactions, and those steps will be elucidated in those specific applications.

#### 2.2.1 Forces Acting on Objects

In many situations, the forces imposed on the object by the fluid can have profound effects on the object including not only position (trajectory), velocity, and acceleration, of course, but also lift, and drag. These can factor into and affect the design of objects such as airplanes and other vehicles. These forces could also be the stress acting on an object like an off-shore structure, or cylindrical support, in the ocean or free-falling objects in a fluid.

Consider a point on an object with surface $\Gamma$. The fluid exerts a surface force per unit area given by

$$ f = \sigma \cdot n \quad \text{on } \Gamma, $$

where $\sigma$ is the stress tensor introduced in Equation (2.3), and $n$ is the unit vector normal to the surface $\Gamma$. In matrix form, the stress $\sigma$ is given by the equation

$$ \sigma = \begin{bmatrix} (-p + 2\mu \frac{\partial u_1}{\partial x_1}) & \mu \left( \frac{\partial u_2}{\partial x_1} + \frac{\partial u_1}{\partial x_2} \right) & \mu \left( \frac{\partial u_3}{\partial x_1} + \frac{\partial u_1}{\partial x_3} \right) \\ \mu \left( \frac{\partial u_1}{\partial x_1} + \frac{\partial u_2}{\partial x_2} \right) & (-p + 2\mu \frac{\partial u_2}{\partial x_2}) & \mu \left( \frac{\partial u_3}{\partial x_2} + \frac{\partial u_2}{\partial x_3} \right) \\ \mu \left( \frac{\partial u_1}{\partial x_1} + \frac{\partial u_3}{\partial x_2} \right) & \mu \left( \frac{\partial u_3}{\partial x_2} + \frac{\partial u_3}{\partial x_3} \right) & (-p + 2\mu \frac{\partial u_3}{\partial x_3}) \end{bmatrix}, $$

(2.7)
where \( u_1, u_2, \) and \( u_3 \) are the \( x_1, x_2, \) and \( x_3 \) components of the velocity vector. Now I can define the \( x_1, x_2, \) and \( x_3 \) components of the force vector per unit area using this definition of stress:

\[
\begin{align*}
f_1 &= \left( 2\mu \frac{\partial u_1}{\partial x_1} - p \right) n_1 + \mu \left( \frac{\partial u_2}{\partial x_1} + \frac{\partial u_1}{\partial x_2} \right) n_2 + \mu \left( \frac{\partial u_3}{\partial x_1} + \frac{\partial u_1}{\partial x_3} \right) n_3, \\
f_2 &= \left( 2\mu \frac{\partial u_2}{\partial x_2} - p \right) n_2 + \mu \left( \frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1} \right) n_1 + \mu \left( \frac{\partial u_3}{\partial x_2} + \frac{\partial u_2}{\partial x_3} \right) n_3, \\
f_3 &= \left( 2\mu \frac{\partial u_3}{\partial x_3} - p \right) n_3 + \mu \left( \frac{\partial u_1}{\partial x_3} + \frac{\partial u_3}{\partial x_1} \right) n_1 + \mu \left( \frac{\partial u_2}{\partial x_3} + \frac{\partial u_3}{\partial x_2} \right) n_2.
\end{align*}
\]

(2.9) \hspace{1cm} (2.10) \hspace{1cm} (2.11)

However, the stress tensor only identifies the pressure and viscous forces acting on the object. These are not the only forces acting on the object. There is also a force exerted on it by gravity,

\[
f_g = mg,
\]

(2.12)

where \( m \) is the mass of the object and \( g \) is the acceleration due to gravity. This force acts at the center of gravity of the object in question. Due to the hydrostatic pressure of the fluid, there is also a buoyant force exerted on the object by the fluid. Per unit area, this force is represented by

\[
f_h = \rho g z,
\]

(2.13)

where \( \rho \) is the density of the fluid, \( g \) is the acceleration of gravity, and \( z \) is the depth or height of the point. To find the total force on the object, I simply integrate these forces over the entire surface of the object. This integral equation is defined by

\[
F = \int_{\Gamma} f_t d\Gamma,
\]

(2.14)

where \( f_t \) is the total force on the object. Analogously, the total moment vector \( M_{x_0} \) about the point \( X_0 \) is

\[
M_{x_0} = \int_{\Gamma} m d\Gamma,
\]

(2.15)
where
\[ m = (x - x_0) \times f(x), \quad x \in \Gamma, \]  
for each individual force.

### 2.2.2 Particle Dynamics

As aforementioned, there are fluid flow simulations in which objects, or particles, move around freely, such as a paratrooper in free fall. The trajectory and movements of the particles affect the flow, and the flow, in turn affects the particles. This coupling continues throughout the trajectory of the particles. Following Johnson’s implementation [18], the motion of these particles under the fluid forces follow Newton’s second law:

\[ F = ma, \]  
(2.17)

Analogously, their rotational motions is governed by a similar equation,

\[ M = Iw, \]  
(2.18)

where \( M \) is the moment vector, \( I \) is the moment of inertia matrix and \( w \) is the angular acceleration.

I can congregate these into one aggregate equation if I let \( F \) now represent the aggregate force and moment vector, \( M \) be the aggregate mass and moment of inertia matrix, and \( A \) represent the composite linear and angular acceleration vector. The equations follow:

\[ F = MA, \]  
(2.19)

These composite vectors and matrices can then be decomposed:

\[
\begin{align*}
F &= \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ M_1 \\ M_2 \\ M_3 \end{bmatrix}, \\
M &= \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{11} & I_{12} & I_{13} \\ 0 & 0 & 0 & I_{21} & I_{22} & I_{23} \\ 0 & 0 & 0 & I_{31} & I_{32} & I_{33} \end{bmatrix}, \\
A &= \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ \Theta_1 \\ \Theta_2 \\ \Theta_3 \end{bmatrix}.
\end{align*}
\]  
(2.20)
The forces $F_1$, $F_2$, and $F_3$ are those imposed on the particle by the fluid. Likewise, the moments $M_1$, $M_2$, and $M_3$ are also exerted by the fluid on the particle. In the matrix $M$, $m$ is the mass of the particle. The elements $I_{11}, \ldots, I_{33}$ are the mass moments of inertia of the object. For example, for a sphere, $I_{11} = I_{22} = I_{33} = I$, and all other moments of inertia are zero. In the vector $A$, the components $A_1$, $A_2$, and $A_3$ are the three components of the acceleration of the particle, and $\Theta_1$, $\Theta_2$, and $\Theta_3$ are the angular accelerations of the particle. For an alternate formulation of particle dynamics, in 2D, with various simulative examples with the 2D formulation, refer to [19].
Chapter 3

Finite Element Formulations

In this chapter, I describe the finite element formulations for the governing equations for the fluid. The semi-discrete formulation of the equations is shown, as it was used a small number of times for space-time code and results validation. The main fluid dynamics solver for the simulations presented in Chapters 4–8 is based on the DSD/SST formulation, which is presented in Section 3.2. All terms including stabilization and nonlinear terms are explicated, and the equations of particle dynamics are expounded to demonstrate the treatment of particle movement.

3.1 Semi–Discrete Formulation

In the semi–discrete formulation, I discretize the spatial coordinates alone. The temporal coordinates are handled with a finite difference approach. The formulation presented here can be used for simulations where there is no mesh movement taking place. This will save on computations and memory compared with the space–time formulation to be presented in Section 3.2. Alternatively, mesh movement can be accounted by added new terms into the formulation, and some methods do this.

To construct the finite element function spaces for the semi–discrete method, the time interval $(0,T)$ is partitioned into discrete time levels $t_n$, where $t_n$ belong to an ordered series of time levels $0 = t_0 < t_1 < \cdots < t_N = T$. The following finite element
interpolation function spaces are defined for each time level $n$: 

$$
(S_u^h)_n = \{ u^h | u^h \in [H^1(\Omega_n)]^{nd}, u^h = g^h \text{ on } (\Gamma_n)_g \}, \tag{3.1}
$$

$$
(Y_u^h)_n = \{ w^h | w^h \in [H^1(\Omega_n)]^{nd}, w^h = 0 \text{ on } (\Gamma_n)_g \}, \tag{3.2}
$$

$$
(S_p^h)_n = (Y_p^h)_n = \{ q^h | q^h \in H^1(\Omega_n) \}. \tag{3.3}
$$

Here $H^1(\Omega_n)$ is the finite-dimensional function space over the spatial domain $\Omega_n$. First-order polynomials are used to form this space over the element domain.

The stabilized, semi–discrete formulation is then written as follows: find $u^h \in (S_u^h)_n$ and $p^h \in (S_p^h)_n$ such that $\forall w^h \in (Y_u^h)_n$ and $q^h \in (Y_p^h)_n$

$$
\int_{\Omega_n} w^h \cdot \rho \left( u^h + u^h \cdot \nabla u^h + f^h \right) d\Omega + \int_{\Omega_n} \epsilon(w^h) : \sigma(p^h, u^h) d\Omega
$$

$$
- \int_{(\Gamma_n)_h} w^h \cdot h^h d\Gamma + \int_{\Omega_n} q^h \nabla \cdot u^h d\Omega
$$

$$
+ \sum_{e=1}^{n_{el,n}} \int_{\Omega^e_n} \frac{\tau}{\rho} \left[ \rho u^h \cdot \nabla w^h - \nabla \cdot \sigma(q^h, w^h) \right].
$$

$$
[\rho \left( u^h + u^h \cdot \nabla u^h + f^h \right) - \nabla \cdot \sigma(p^h, u^h)] d\Omega + \sum_{e=1}^{n_{el,n}} \int_{\Gamma^e_h} \delta \nabla \cdot w^h \rho \nabla \cdot u^h d\Omega = 0. \tag{3.4}
$$

This process is applied sequentially to all time levels $\Omega_0, \Omega_1, \Omega_2, \ldots, \Omega_{N-1}$.

### 3.1.1 Interpretation of the Terms

In the variational formulation given by Eq. 3.4, the first four terms on the left-hand–side constitute the Galerkin formulation of the problem. The fifth and sixth terms are the needed stabilization terms; otherwise, oscillations occur in advection–dominated flows. These terms are defined in Section 3.4. They share the same definitions for both the space–time and semi–discrete formulations.
3.1.2 Time Derivatives

In the semi–discrete formulation, the time derivative is handled in a finite–difference fashion. The derivative term \( \dot{u}^h \) is as follows:

\[
\dot{u}^h = \frac{u^h_{n+1} - u^h_n}{\Delta t}.
\]  

(3.5)

With this derivative definition, \( u^h \) and \( f^h \) decompose into the new forms,

\[
u^h = (1 - \alpha)u^h_n + \alpha u^h_{n+1},
\]  

(3.6)

\[
f^h = (1 - \alpha)f^h_n + \alpha f^h_{n+1}.
\]  

(3.7)

The decomposition in Equation 3.6 is valid for linear terms. But special care must be taken when using this decomposition with non-linear terms. I solve the entire semi–discrete formulation, Equation 3.4, for \( u^h_{n+1} \); \( u^h_n \) is given and known. Unknowns only exist at time level \( n + 1 \). The computation is started with

\[
u^h = u_0.
\]  

(3.8)

In Equation 3.4, all pressure degrees of freedom come from the \( n + 1 \) level, only. And the decomposition Equation 3.6 does not apply to all terms in Equation 3.4. The fourth and the sixth terms (the velocity in the mass balance equations) are simply \( u^h_{n+1} \).

The \( \alpha \) parameter governs and controls the type of numerical scheme applied to the semi–discrete formulation inasmuch as it defines my decomposition scheme in Equations 3.6 and 3.7. The choices follow.

\( \alpha = 0.0 \): Forward Difference

\( \alpha = 0.5 \): Central Difference or Trapezoidal Rule

\( \alpha = 1.0 \): Backward Difference
The Forward Difference scheme is used if an explicit method is desired since all the unknowns will be at the level \( n \) which are the known quantities. The Central Difference scheme is attractive if an accurate solution is desired. It is second-order accurate in time. The Backward Difference scheme is used, primarily, to obtain steady state solutions. This is done by setting the time step to be very large with that particular choice of \( \alpha \).

3.2 Space-Time Formulation

To handle the time-variant spatial domains encountered in aerodynamic separation problems, the deforming-spatial-domain/stabilized space-time (DSD/SST) finite element formulation is employed. This method has been applied to a large number of problems with moving boundaries and interfaces with effective results, and is well suited to handle the time-variant spatial domains. In this formulation, the temporal coordinate, along with the spatial coordinate, is interpolated using the finite element basis functions.

This formulation is similar to the Arbitrary Lagrangian–Eulerian formulation (ALE) which handles mixed Lagrangian and Eulerian descriptions of the problem, first introduced with finite difference and finite volume methods [4–7] and then eventually with finite element methods [8–11]. This method can be used where some parts of the domain require an Eulerian description since the mesh does not move, or with parts of the domain when a Lagrangian description is required because the mesh is deforming, or even when a mixed description is desired when the motion of the mesh is arbitrary. The following space-time formulation was first introduced in [2, 3], and has also been applied to compressible flows, as well, in [12, 20].

In order to construct the finite element function spaces for the space-time method, I must partition the time interval \((0, T)\) into subintervals \( I_n = (t_n, t_{n+1}) \), where \( t_n \) and \( t_{n+1} \) belong to an ordered series of time levels \( 0 = t_0 < t_1 < \cdots < t_N = T \). Letting \( \Omega_n = \Omega_{t_n} \) and \( \Gamma_n = \Gamma_{t_n} \), the space-time slab \( Q_n \) is defined as the domain enclosed by the surfaces \( \Omega_n, \Omega_{n+1}, \) and \( P_n \), where \( P_n \) is the surface inscribed by the
boundary $\Gamma_t$ as $t$ traverses $I_n$. The space–time concept is depicted in Figure 3.1 for the space–time slab, $Q_n$. The surface $P_n$ is decomposed into $(P_n)_g$ and $(P_n)_h$ with respect to the type of boundary condition (Dirichlet or Neumann, respectively) being imposed. For each space–time slab, I define the corresponding finite element function spaces $(S^h_u)_n$, $(V^h_u)_n$, $(S^h_p)_n$, and $(V^h_p)_n$ as follows:

\begin{align}
(S^h_u)_n &= \{ u^h | u^h \in [H^{1h}(Q_n)]^{n+2}, u^h = g^h \text{ on } (P_n)_g \}, \\
(V^h_u)_n &= \{ w^h | w^h \in [H^{1h}(Q_n)]^{n+2}, w^h = 0 \text{ on } (P_n)_g \}, \\
(S^h_p)_n &= (V^h_p)_n = \{ q^h | q^h \in H^{1h}(Q_n) \}.
\end{align}

Here $H^{1h}(Q_n)$ is the finite-dimensional function space over the space–time slab $Q_n$. This time, in the space–time formulation, first–order polynomials in both space and time are used to form the space over the element domain. The interpolation functions, however, are continuous in space but discontinuous in time.

The stabilized space–time formulation for deforming domains is now written as follows: given $(u^h)_n$, find $u^h \in (S^h_u)_n$ and $p^h \in (S^h_p)_n$ such that $\forall w^h \in (V^h_u)_n$ and
\[ q^h \in (V_p^h)_n \]

\[
\int_{Q_n} w^h \cdot \rho \left( \frac{\partial u^h}{\partial t} + u^h \cdot \nabla u^h + f^h \right) dQ + \int_{Q_n} \epsilon(w^h) : \sigma(p^h, u^h)dQ
\]

\[
+ \int_{Q_n} q^h \nabla \cdot u^h dQ + \sum_{e=1}^{n_{el}} \frac{\tau}{\rho} \left[ \rho \left( \frac{\partial w^h}{\partial t} + u^h \cdot \nabla w^h \right) - \nabla \cdot \sigma(q^h, w^h) \right] dQ
\]

\[
\left[ \rho \left( \frac{\partial u^h}{\partial t} + u^h \cdot \nabla u^h + f^h \right) - \nabla \cdot \sigma(p^h, u^h) \right] dQ + \sum_{e=1}^{n_{el}} \int_{Q_n} \delta \nabla \cdot w^h \rho \nabla \cdot u^h dQ
\]

\[
+ \int_{\Omega_n} (w^h)^+ \cdot \rho \left( (u^h)^+ - (u^h)^- \right) d\Omega = \int_{(P_n)^h} w^h \cdot h^h dP. \quad (3.12)
\]

This process is applied sequentially to all the space–time slabs \( Q_0, Q_1, Q_2, \ldots, Q_{N-1} \).

The explanation of certain notation in Equation 3.12 follows:

\[
(u^h)^\pm_n = \lim_{\epsilon \to 0} u(t_n \pm \epsilon), \quad (3.13)
\]

\[
\int_{Q_n} (\cdots) dQ = \int_{I_n} \int_{\Omega_n} (\cdots) d\Omega dt P. \quad (3.14)
\]

\[
\int_{P_n} (\cdots) dP = \int_{I_n} \int_{\Gamma_n} (\cdots) d\Gamma dt P. \quad (3.15)
\]

The computations start with

\[
(u^h)^-_0 = u_0. \quad (3.16)
\]

### 3.2.1 Interpretation of the Terms

In the variational formulation given by Eq. 3.12, the first three terms, the sixth term, and the right–hand–side comprise the Galerkin formulation of my problem, the momentum balance equation and the mass balance equation. The sixth term weakly enforces continuity of the velocity field across the space–time slabs, since the
interpolation functions are discontinuous in time, and since I solve the equation one space–time slab at a time. Due to numerical instabilities that occur in advection-dominated flows and from oscillations that occur when different combinations of interpolation functions for velocity and pressure are used, I include stabilizing terms. The first series of element–level integrals in Eq. 3.12 are least–squares terms based on the momentum equation. This term stabilizes the standard Galerkin form. The second series of element–level integrals are added to the formulation for numerical stability at high Reynolds numbers. These are least–squares terms based on the continuity equation. The stabilization coefficients $\tau$ and $\delta$ are defined at the element level. Both stabilization terms are weighted residuals, and therefore maintain the consistency of the formulation. For an exact solution, they converge to zero.

3.2.2 Space–Time Elements

Time adds an extra dimension to the normal spatial elements in the space–time formulation. And, thus, the elements become space–time elements. Here I use a standard spatial mesh, and allow the solver to automate the space–time aspect. And so, the space–time elements in my space–time simulations are simply extensions of the elements in the spatial mesh. The spatial mesh at the bottom of the space–time slab is simply connected to the deformed mesh at the top of the space–time slab. In cases where no movement occurs, both meshes, at the top and bottom of the space–time slab, are identical. Definitely, the connectivity remains the same, regardless of deformation.

In 2D, a quadrilateral (4–noded) element becomes a rectangular prism, or a hexahedral element, with time as the third dimension. A triangle becomes a triangular prism, or a wedge element. The thickness of the space–time element is the time step size. And since the spatial meshes and nodes at each level of a space–time slab have the same temporal coordinate, the algebra and computations are simplified.
3.3 Nonlinearity

In the conservation of momentum Equation 2.1, the advection term is nonlinear. Since I employ linear solvers on the equations, a suitable method must be used to linearize the problem. The implementation of choice in the succeeding simulations was the Newton-Rhapson method. First, I define the nonlinear term

\[ N(u) \equiv u \cdot \nabla u, \]

which is a nonlinear function of \( u \). I then must dissect this term into known and unknown parts and solve for the unknown iteratively. This decomposition yields

\[ u = u_i + \Delta u, \]

where \( i \) is the nonlinear iteration counter, and \( \Delta u \) is the increment of my unknown vector \( u \), at iteration \( i \). The term \( \Delta u \) becomes my new unknown. And I expand the nonlinear term with the Taylor series producing

\[ N(u_i + \Delta u) = N(u_i) + \left. \frac{\delta N}{\delta u} \right|_{u_i} \Delta u + O(\Delta u_2). \]

Now the equation becomes linear through this decomposition. I can now solve the finite-element formulation iteratively in this incremental form revealing that

\[ N(u) = u_i \cdot \nabla u_i + u_i \cdot \nabla \delta u + \delta u \cdot \nabla u_i + \delta u \cdot \nabla \delta u. \]

Realizing that the last term in Equation 3.20 corresponds to a part of the last term in Equation 3.19, which I do not wish to calculate assuming it is small, I ignore it. Ignoring the higher-order terms, the nonlinear term now becomes

\[ N(u) = u_i \cdot \nabla u_i + u_i \cdot \nabla \delta u + \delta u \cdot \nabla u_i. \]

In the aforementioned finite element variational formulae, nonlinear terms also appear in the stabilization parameters. Using a more simple implementation, I do not calculate every term in Equation 3.21. Instead, to simplify, I use

\[ N(u) = u_i \cdot \nabla u_i + u_i \cdot \nabla \delta u. \]
Additionally, the velocity quantities comprising the stabilization terms \( \tau \) and \( \delta \) are retrieved from the previous iteration \( u_i \). This method is called a modified Newton Rhapson method since I do not apply Equation 3.21 to all nonlinear terms.

### 3.4 Stabilization

The stabilization term \( \tau \), found in Sections 3.2 and 3.1, is defined below:

\[
\tau = \left[ \left( \frac{2}{\Delta t} \right)^2 + \left( \frac{2\|u^h\|}{h_e} \right)^2 + \left( \frac{4\nu}{h_e^2} \right)^2 \right]^{\frac{1}{2}},
\]

where \( \nu \) is the kinematic viscosity, \( \Delta t \) is the time step, and \( h_e \) is a suitable choice for the element length. All terms are individually chosen for each element. In the following simulations, the maximum edge length is used as the measure of element length. Better measures of element length, based on the direction of advection, can be found in [21].

The definition of the stabilization term \( \delta \) is given by:

\[
\delta = \frac{h_e}{2} \|u^h\| z,
\]

where \( z \) is defined as

\[
z = \begin{cases} 
\left( \frac{Re_u}{3} \right) & Re_u \leq 3 \\
1 & Re_u \geq 3
\end{cases}.
\]

I define \( Re_u \) as the element Reynolds number according to:

\[
Re_u = \frac{\|u^h\| h_e}{2\nu}.
\]

Further discussions of these stabilization terms, including their derivations, can be found in [22,23].

### 3.5 Implementing Particle Dynamics

In Section 2.2.2, the governing equations of particle dynamics were presented. Now I will expound on the process of solving the equations based on Johnson’s original
code [18]. The unknowns in particle dynamics are located at both the top and bottom of the space–time slab in the space–time element formulation.

In Equation 2.17, the three equations for the force are decoupled from each other; although, the three equations for the moments are not decoupled from each other. Considering the first decoupled equation, I find

\[ F_1 = mA_1. \] (3.27)

Since I am solving for an entire space–time slab at once, for the force, I choose the value of the force imposed on the particle in the middle of the slab in the temporal direction. Setting \( F_1^n \) as the force at the lower temporal level of the time slab and \( F_1^{n+1} \) as the force at the upper level, then

\[ F_1 = \frac{F_1^n + F_1^{n+1}}{2}. \] (3.28)

Then I can discretize the particle’s acceleration, as well,

\[ A_1 = \frac{V_1^{n+1} + V_1^n}{\delta t}, \] (3.29)

where \( \delta t \) is the time step. Dropping the acceleration term and using velocity, I can then rewrite Equations 3.27 as

\[ \frac{F_1^{n+1} + F_1^n}{2} = m \frac{V_1^{n+1} + V_1^n}{\delta t}. \] (3.30)

The forces are computed from the flow field at each time level \( n \) and \( n+1 \). Therefore they are known quantities. And since the velocity of the particle is assumed to be continuous across the time slab, I know \( V_1^n \). It is equivalent to \( V_1^{n+1} \) calculated for the space–time slab from \( n - 1 \) to \( n \). If I rearrange Equation 3.30, I can solve for the unknown,

\[ V_1^{n+1} = V_1^n + \frac{\delta t}{2m} (F_1^{n+1} + F_1^n), \] (3.31)

where \( V_1^{n+1} \) is the unknown at each nonlinear iteration and governs the update of the particle position and, consequently, the mesh coordinates.
I, then, continue to work reversely to obtain the particle's position,

\[ \dot{X}_1 = V_1. \]  

(3.32)

Analogously, I can expand \( \dot{X}_1 \) as I did \( \dot{V}_1 \) in Equation 3.29, and \( V_1 \) as I did to \( F_1 \) in Equation 3.28. Again, rearranging for my unknown, I arrive at

\[ X_1^{n+1} = X_1^n + \frac{\delta t}{2} (V_1^{n+1} + V_1^n), \]

(3.33)

Again, the particle's coordinates are continuous across the space–time slab. So \( X_1^n \) is known. So Equations 3.33 and 3.31 provide the required movement of the particle based on the fluid forces acting upon it. In cases where there are other external forces acting on the particle, like gravity, those forces are added into the force vector along with the computed fluid forces.
Chapter 4

Mesh Generation and Update

The trajectories of a jumping paratrooper or plummeting cargo from a military cargo aircraft can be wildly dappled and somewhat unpredictable depending on the conditions. As mentioned previously, the DSD/SST procedure is well suited to simulate these separation problems since the ability to handle deforming boundaries is part of the formulation; it is a built-in feature. However, the complex and deforming geometries encountered in paratrooper exit, cargo deployment, and general air decelerator systems introduce some additional challenges in the areas of mesh generation, mesh moving and remeshing. And these all play a crucial role in Fluid Object Interaction simulations. In this chapter, I will explore challenges associated with mesh generation and update, identify new techniques to handle boundary and domain deformations, and solve the problems of mesh moving, distortion, remeshing, and projection.

Section 4.1 describes the previous work in mesh generation and the mesh generation strategy and software used in modeling the paratrooper, aircraft, and cargo payload geometries involved in the Fluid Object Interaction simulations presented in this thesis. A brief summary is supplied regarding my new work in mesh generation, specifically boundary layer mesh generation. In DSD/SST simulations, it is not desirable or economical to remesh at each time step, so sophisticated mesh moving algorithms are needed to handle motion of the separating objects like the
jumping paratrooper. In Section 4.2, I present previously developed, yet currently utilized techniques for handling mesh moving culminating in the automatic handling of the deformation of the mesh. In Section 4.3, my new contributions in mesh moving strategies are demonstrated for two 3-D examples, and the different strategies are assessed. As these simulations progress with automatic mesh moving, the mesh quality deteriorates. Once this goes beyond an acceptable limit, the fluid domain is remeshed. Section 4.4 describes previous work I follow in remeshing procedures, while section 4.5 illuminates the subsequent projection strategies used starting with previous projection tactics ending with my new successful projection tactics and new divergence-free projection method. Together, both allow the Fluid Object Interaction simulations to continue with new meshes.

4.1 Mesh Generation

This section, as with other sections, is divided to show previous work that I currently use. Then I explicate what new work I have undertaken and completed in that area. I commence with mesh generation.

4.1.1 Previous Work

Large finite element meshes are needed to accurately represent complex aircraft and paratrooper geometries. Many times, I need to have meshes with high levels of refinement in regions like the boundary layer of the aircraft to ensure the proper drag on the exiting cargo and falling paratrooper. However, that same refinement in other areas might be wasteful. Therefore, it is desirable to limit refinement in those key regions, keeping mesh sizes manageable. Also, meshes with elements of good quality are desirable to oppose the onset of severe mesh element distortion and entanglement delivering the highest accuracy in numerical FD solutions in our FOI simulations. Unstructured mesh generation techniques are exploited to generate high quality meshes with controlled levels of refinement. I do not describe the automatic
mesh generation tools used in the simulations in this thesis because they are very well developed. I use a mesh generation software based on the strategies described in [18] and [24]. This software is well suited for handling the complex geometries of cargo aircraft, cargo payloads, and bulky paratroopers laden with ammunition and supplies.

4.1.2 New Contributions

My new contributions to the area of mesh generation are in the focus of boundary layer mesh generation. I have deferred the expansion of this topic until Chapter 6, Section 6.1, which more pertinently talks about boundary layer resolution. All normal mesh generation was performed as stated in Section 4.1.1. All boundary layer mesh generation is my contribution.

4.2 Mesh Moving Overview: Previous Work

Mesh-moving strategies deal both with the interior of the mesh and the boundaries of the mesh. In moving the boundaries, one finds that the motion of the mesh is just a requirement, or the result of the fluid boundary condition. On solid surfaces or boundaries, I require that no flow cross the boundary. This is seen in

\[ \mathbf{v} \cdot \mathbf{n} = \mathbf{u} \cdot \mathbf{n} \quad \text{on } \Gamma_t, \]

\[ \mathbf{v} \cdot \mathbf{t} = \text{arbitrary} \quad \text{on } \Gamma_t, \]

where \( \mathbf{v} \) is the mesh velocity and \( \mathbf{u} \) is the fluid velocity. Three choices remain for the mesh velocity at the boundary.

**Known:** When the motion of the object is prescribed, this is the utilized option. In these problems, the boundary condition sets the fluid velocity on the boundary or interface; therefore, the mesh velocity is also known. Consequently, mesh update must take place every time step.
**Unknown:** This is the option used in a fluid-interface or free surface-type problem. Here the mesh velocity is not a set boundary condition on the fluid; instead, it is in direct response to the unknown fluid velocity at the interface and is, therefore, unknown, as well. The fluid velocity results form the solution of the governing equations and changes slightly with each nonlinear iteration causing the mesh velocity to change with each nonlinear iteration.

**Indirect:** This is the option explored in the Fluid Object Interaction simulations presented here. The fluid velocity is solved for, first. This then creates forces that affect structures or objects in the simulation. As they move, the mesh is moved. This is seen with the falling paratrooper. At each nonlinear iteration the aerodynamic forces acting on him are found. Those forces cause a certain displacement and trajectory. This displacement causes the mesh to move, coupling the fluid and solid dynamics together.

Equations 4.1 and 4.2 handle the boundaries of the domain. In the interior of the mesh domain, mesh-moving strategies that can handle large deformations with minimal loss in mesh quality are sorely needed. For some specific problems, special purpose mesh moving algorithms can be designed and are effective. These methods benefit from being more efficient and controllable than fully automatic methods. Algebraic mesh generation and mesh moving strategies have been successfully demonstrated on the translational and pitching motions of an airfoil [25, 19]. The same success was reached on the motions of ram-air parachute systems [26]. I will briefly expound on these strategies and the methodology. Still, these approaches are limited to the class of problems for which they were developed. They are not helpful for problems where the behaviors are not sufficiently predictable. Also, these approaches are difficult to extend to problems involving complex geometries, especially in 3-D.

The falling trajectories of jumping paratroopers can be very erratic and varied. I therefore employ automatic and unstructured mesh generation and automatic mesh moving schemes. This is also done due to the complex 3-D geometries of both the paratrooper and cargo aircraft. These schemes introduce an increased computational
cost since they involve automatic 3-D mesh generation and require the solution of an additional system of equations for the mesh motion. However, these methods are well suited for handling complex geometries and arbitrary motions for this class of FOI problems. Previously, automatic mesh moving methods have been developed and effectively demonstrated for problems using space–time finite element formulations [25, 14, 27, 28, 22, 23, 29].

Again, separating objects such as exiting cargo or jumping paratroopers can cause severe deformation to a mesh when using automatic mesh moving. I break down the motion of the objects into three groups.

- **Rigid–body translation:** In most cases and in the case of a jumping paratrooper or falling cargo payload, most of the motion is translational. This type of motion is the easiest to handle. One way to handle it is to globally move the entire mesh with the rigid–body motion. Sometimes this cannot be done, as in a case when a jumping paratrooper falls away from another object—relative motion between objects. In these cases, I use automatic mesh moving schemes to handle the motion, which will come with some mesh deformation.

- **Rigid–body rotation:** As the paratrooper falls and translates, he often rotates about his center of gravity. Paratroopers can experience rotational motions not only during the initial jump, but throughout the entire time period which I will examine, approximately the first 2 seconds. Spinning, diving, and tumbling are usual motions observed from paratrooper jumps as each individual jump is highly unique. And these rotational motions are more susceptible to mesh tangling than pure translational motions.

- **Deformations:** The greatest challenge in mesh moving is deformational motion. Deformations are easily observed in simulations involving Fluid Structure Interactions, such as the simulation of an inflating parachute as it falls and rotates in the sky. In the problems presented in this thesis, deformations will not be considered. Objects such as cargo payloads and falling paratroopers are treated
as rigid-bodies that cannot deform. Incidentally, deformations are even more subject to distortion and entanglement.

In Sections 4.2.1–4.2.4, different traditional, 2D–initiated mesh moving strategies are remonstrated [18] as they culminate in the treatment of the mesh as a pseudo–elastic solid [30] enabling automatic mesh moving for the paratrooper and cargo applications. Then, in Section 4.3, previously explored 3–D mesh moving tactics along with my own new mesh moving tactics are examined and exploited to maintain accuracy, mesh quality, and a good solution. Remeshing and new projection methods are finally discussed in Sections 4.4 and 4.5.

### 4.2.1 One Remesh Per Time Step

One possible method for handling moving boundaries and interfaces is to remesh every time step with the help of an automatic mesh generator. After each time step (Figure 4.1), a new mesh will be generated for the current configuration of the domain which includes the current position of all objects such as deployed payloads.

The solution from the old mesh, during the last time step, is then projected to the new mesh. This step of projection, however, does introduce errors. That is not what makes this method untenable for our cases. It is the fact that this method does not work well with large, complex 3D domains. The costs are too great. I, instead,
would like to limit the costs from remeshing, and, therefore, limit remeshing. Moreover, there is a parallel implementational cost associated with this method because I must map the finite element mesh on to parallel processors (this includes ordering, reordering, shifting of nodes and elements) and compute the network of communication links between all elements and processors. In this method these costs occur every time step. This method has been used in 2D [25, 23].

4.2.2 Specialized Design

A second method for handling moving boundaries and interfaces is to create a special mesh suited to the problem. In this case, a mesh generator is created according to the type of mesh motion to be simulated or observed. Yes, there is a new mesh at every time step, but first, the costs are algebraic, so it is not a deterrent. Secondly, a new mesh is not literally regenerated; I do not remesh. The connectivity of the nodes remains exactly the same, although the distortion and location of elements and nodes may change (shown in Figure 4.2).

Each node at one time step can be mapped to another node at the previous time step. And therefore, in this method, I lose the parallelization setup costs associated with remeshing at every time step. I can even utilize this method in 3D simulations. However, as the 3D domain and geometry become increasingly complex, the level of
difficulty in creating these problem–dependent mesh generators also increases. Additionally, the mesh generator created for these problems are very problem–specific and cannot be used for many problems.

An example of a special mesh design is a simulation I performed on a ball in shear flow. Shown in Figure 4.3, the inner domain is split into two parts. The first is an unstructured fixed spherical zone around the ball. The zone moves with the ball to maintain the same refinement about the ball no matter its speed or location. Outside the fixed spherical zone surrounding the ball, the mesh is moving, and the elements are allowed to deform (there is also a third, outer zone which will be discussed in Section 4.2.4).

### 4.2.3 Automatic Mesh Moving: Linearly Elastic Model and Formulation

This next mesh moving scheme allows me to handle the deformation of the mesh automatically based on the deformation of the boundary. The solid mesh is treated as a pseudo–solid, and the equations of linearly elasticity are used to solve for the displacements of the internal node based on the boundary deformation conditions. Now the movement of the mesh can be modeled or prescribed in any way I choose. But the equations of linear elasticity function adequately for a generalized problem. Our solid is a pseudo–solid because the characteristics of the solid will be altered to suit the individual problem and expedite mesh moving.

There are no mesh generation costs in this method. The only restriction is that the deformation of the mesh be small to eschew unacceptable levels of distortion. The topology, or connectivity, of the mesh is constant since the nodes are only being moved around. And with this method, used alone, there are no projection overhead costs from parallelizational setup and no introduction of projection errors. The only drawback is the higher costs of solving a new equation for the linear elastic model.

The corresponding equations, governing this mesh motion, follow. Let $\Omega^m \subset \mathbb{R}^{m-d}$ be the spatial domain bounded by $\Gamma^m$. The boundary $\Gamma^m$ is composed of $(\Gamma^m)_p$ and
Ball Simulation: Fixed Spherical Inner Zone Surrounding Ball

Ball Simulation: Fixed Inner Zone with Moving Outer Zone

Figure 4.3. Ball in Shear Flow: 3D Surface Meshes.

\[(\Gamma^m)_h,\] ostensibly corresponding to the Dirichlet–type and Neumann–type boundary conditions. To clarify, the superscript “m” corresponds to the mesh moving equations. The equations of motion for the pseudo–solid are:

\[\nabla \cdot \sigma^m + \Gamma^m = 0 \text{ on } \Omega^m, \tag{4.3}\]
where $f^m$ is an external forcing function (which can be used to control mesh moving properties or represent actual external forces) and $\sigma^m$ is the Cauchy stress tensor for the pseudo–solid. For a linear–elastic solid, $\sigma^m$ is defined, using the strain tensor $\varepsilon^m$, as

$$\sigma^m = \lambda^m \text{tr}(\varepsilon^m) I + 2\mu^m \varepsilon^m$$  \hspace{1cm} (4.4)

where $\varepsilon^m$ is defined as

$$\varepsilon^m = \frac{1}{2} \left( \nabla y^m + (\nabla y^m)^T \right).$$  \hspace{1cm} (4.5)

Here $\lambda^m$ and $\mu^m$ are elastic material constants, and $y^m$ is the aggregate material displacement.

The Dirichlet–type and Neumann–type boundary conditions are represented as

$$y^m = g^m \text{ on } \Gamma_g^m,$$

$$n \cdot \sigma^m = h^m \text{ on } \Gamma_h^m.$$  \hspace{1cm} (4.6)

Now, function spaces must be formed to set up the finite element formulation. For the mesh domain, $\Omega^m$, the corresponding finite element function spaces $S^h$ and $V^h$ are defined as follows:

$$S^h = \{ (y^m)^h | (y^m)^h \in [H^{1h}(\Omega^m)]^{n_{sd}}, (y^m)^h \equiv (g^m)^h \text{ on } \Gamma_g^m \},$$  \hspace{1cm} (4.7)

$$V^h = \{ w^h | w^h \in [H^{1h}(\Omega^m)]^{n_{sd}}, w^h \equiv 0 \text{ on } \Gamma_g^m \}. $$  \hspace{1cm} (4.8)

Here, $H^{1h}(\Omega^m)$ is the finite-dimensional function space over $\Omega^m$. Now, the finite element formulation for the pseudo–solid mesh moving scheme can be written. I want to find $(y^m)^h \in S^h$ such that $\forall w^h \in V^h$

$$\int_{\Omega^m} \varepsilon^m(w^h) : \sigma^m(y^m)^h d\Omega^m - \int_{\Omega^m} w^h : f^m d\Omega = \int_{\Gamma_h^m} w^h : h^m d\Gamma^m.$$  \hspace{1cm} (4.9)

### 4.2.4 Automatic Mesh Moving With Non-moving Zones

The last method to be mentioned for problems with moving boundaries or interfaces is the use of non-moving zones amid automatic mesh moving. This is the
situation that arises when the trajectory of the moving object is somewhat determinable or generally predictable. In other words, this is useful when the bulk of the mesh moving occurs in one place, location, or region of the mesh during the mesh moving simulation.

In these situations, costs can be reduced and minimized if I relegate the mesh moving region from the entire domain to a subset of the mesh that encompasses possible trajectories of the moving object. This will reduce the number of mesh moving equations to be solved resulting in savings of both memory and time.

Care must be taken with this tactic as one might overconstrain the mesh moving during a simulation. If the region is too small and overconstrains the movement of the mesh, you effectively increase the amount and rate of distortion for those moving elements, more than the amount and rate of distortion for those moving elements if moving were allowed to occur for a larger subset of elements. A balance problem results where one must balance the savings in speed and memory from using nonmoving zones and reducing the number of equations, with the increased distortion and time additions of sooner and more frequent remeshing and projection for the smaller moving sections. But usually, if the mesh is large (500,000+ elements) and the moving object moves in a certain region of the mesh, it is a good idea to employ nonmoving zones.

Again, looking at the ball in shear flow in Figure 4.4, a clear section for moving elements can be seen, a section created for this mesh to minimize the mesh moving costs. This particular simulation utilizes both special mesh design (Section 4.2.2) and nonmoving zones. There is no reason why some of these methods cannot be combined with others. This can even results in increased savings.

### 4.3 3–D Mesh Moving Strategies

The solution for the displacement of the pseudo–solid has no physical significance whatsoever. Its sole purpose is to deform the FD mesh to handle motions and deformations from mesh boundaries and interfaces with minimal distortion. Consequently,
Ball Simulation: Fixed Outer Zone with Moving Trajectory Zone

Figure 4.4. Ball in Shear Flow: 3D Surface Mesh.

It is allowable to modify the governing equation for the pseudo-solid to obtain more desirable results for the mesh moving scheme, as long as the boundaries and interfaces move and deform as they should. The internal mesh movement does not carry great import as long as the distortion is low. Various attempts have been made to modify the equations for the pseudo-solid to obtain more desirable mesh moving performance.

In order to assess the effectiveness of the different mesh moving strategies which will be presented, two measures of mesh quality are defined. These two 3-D quality measures were introduced in [14]. The elemental volume deformation ($f_V$), and the
elemental aspect ratio deformation \( f_{AR}^e \) are defined as

\[
f_V^e = \left| \log \left( \frac{V^e}{V_0^e} \right) / \log (2.0) \right|, \tag{4.10}
\]

\[
f_{AR}^e = \left| \log \left( \frac{AR^e}{AR_0^e} \right) / \log (2.0) \right|, \tag{4.11}
\]

where \( AR^e \) is the element aspect ratio. The subscript "o" corresponds to the values in the original undeformed mesh. \( AR^e \) is defined as

\[
AR^e = \frac{(\text{Maximum edge length})^3}{V^e}. \tag{4.12}
\]

For a given mesh, the global volume and aspect ratio deformation factors \( f_V \) and \( f_{AR} \) are defined to be the maximum value of the corresponding elemental deformation factors in the mesh. Therefore the determination of the best time to stop moving the mesh, due to high distortion, and then remesh is based on the maximum value or the one element with the worst deformation according to these definitions. And many methods have been tried to reduce and retard the maximum elemental deformation.

### 4.3.1 Jacobian–based Stiffening: Previous Work

In meshes with various levels of refinement, distortion can do more harm to smaller elements. Different approaches can be taken to stiffen smaller elements to reduce mesh distortion. Our goal is to allow the larger elements to absorb the deformation since they have more volume to be distorted allowing the computation to persist much longer. Following a specific approach introduced in [25] and developed in [30], I utilize a general measure where smaller elements are stiffened through the modification of the Jacobian of the transformation term from \( d\Omega \) in Eq. 4.9. Through discretization, I find that

\[
\int_{\Omega} \ldots d\Omega = \sum_e \int_{V^e} \ldots \det \left( \frac{\partial \mathbf{x}}{\partial \mathbf{x}_i} \right) dV. \tag{4.13}
\]

where \( \mathbf{x} \) are the mesh coordinates, \( \mathbf{x}_i \) are the local element coordinates, and \( V^e \) is the element domain. Here, I employ our general Jacobian–based stiffening approach
by dividing the Jacobian of the transformation in Eq. 4.9 by a non-dimensional value that varies with the magnitude of the Jacobian:

$$\int_{V_e} \cdots \det \left( \frac{\partial \mathbf{x}}{\partial \mathbf{X}_i} \right) dV \mapsto \int_{V_e} \cdots \left( \frac{\det \left( \frac{\partial \mathbf{x}}{\partial \mathbf{X}_1} \right)}{\left( \frac{\det \left( \frac{\partial \mathbf{x}}{\partial \mathbf{X}_i} \right)}{\det \left( \frac{\partial \mathbf{x}}{\partial \mathbf{X}_1} \right)} \right)^a} \right) \chi dV, \quad (4.14)$$

where $\chi$ is the stiffening power and $a$ corresponds to the global average. Thus, the dividend is smaller for smaller elements effectively multiplying the Jacobian term by a larger factor thereby stiffening the element. When $\chi = 0.0$, the dividend is equal to one, and the pseudo-solid is represented by the standard formulation for a homogeneous linear-elastic material. For $\chi \neq 0.0$, the pseudo-material is non-homogeneous.

### 4.3.2 Jacobian–based Stiffening: New Work

In this thesis, I have taken this Jacobian–based stiffening technique, used in 2D [30], and applied it in 3D, specifically to the jumping paratrooper simulation. It should be noted Stein found the optimal value of $\chi$ to be variable depending on the motion.

In Table 4.1, for the simulation of a jumping paratrooper, different values of $\chi$ produce different results. The object of these tests is to determine the optimal value of $\chi$. From the results, I see a value of $\chi = 1.0$, on average, usually allows more mesh moving time steps before a certain distortion limit is reached. For this reason, most computations are run with a value of $\chi = 1.0$. Above that, the results are not conclusive or absolutely certain. Comparing the different cases with different values of $\chi$, there are meshes for which cases with higher values of $\chi$ produce more time steps within that given, same-numbered mesh (time steps per mesh are the time steps from the last remesh to the next while the nodes retain the same connectivity or same mesh). And there are meshes where lower values work better. It is even increasingly difficult to compare the same numbered mesh between different test cases: all meshes
after the first mesh are different for all six cases since Mesh 1 was used to compute for a different number of time steps in each case.

Compounded upon this difficulty is the fact that even though, in Table 4.3, the case with \( \chi = 1 \) had the highest average number of time steps per mesh, the case with \( \chi = 2 \) had the highest total number of time steps. Which is preferable? More total time steps could mean more distortion slowing the time marching process down. With increased distortion comes an increase in remeshing and projection. Projection introduces errors that can slightly alter the path of the paratrooper and greatly change the number of time steps by changing the point where the paratrooper hits the boundaries of the mesh moving domain, or remeshing box. Contrastingly, more time steps could also mean less error which results from less projections and remeshes due to less distortion allowing the paratrooper to hit the remeshing box at closer point. With multiple meanings from increased steps it is difficult to conclusively infer anything from such results.

Noticeably, the cases for \( \chi = 4 \) and \( \chi = 6 \) are missing. Both of these cases became tangled, and the mesh moving process was halted prematurely. Therefore they cannot be compared. Evidently, the higher the value of the \( \chi \), the greater is the tendency and possibility for tangling. It is, however, just a tendency, as the case \( \chi = 5 \) successfully completed the simulation.

### 4.3.3 Variable Stiffening: Previous Work

Another approach for producing variable element stiffness was introduced by Masud [27]. Here, a function \( \tau \) was defined to increase the stiffnesses of smaller elements and thus to preserve mesh quality. This approach can be implemented into the mesh moving scheme by modifying the first term of Eq. 4.9 as

\[
\int_{\Omega} e^m(w^h) : \sigma^m(y^m)^h d\Omega \quad \longrightarrow \quad \int_{\Omega} e^m(w^h) : \sigma^m(y^m)^h (1 + \tau^e) d\Omega,
\]  

where

\[
\tau^e = \frac{V_{max} - V_{min}}{V_e},
\]  

(4.15)  

(4.16)
and $V^e$, $V_{\text{min}}$, and $V_{\text{max}}$ correspond to the volume of element "e", and minimum and maximum element volumes. Thus,

$$V = V_{\text{min}} \rightarrow (1 + \tau^e) = V_{\text{max}} / V_{\text{min}},$$  \hspace{1cm} (4.17)$$

$$V = V_{\text{max}} \rightarrow (1 + \tau^e) = 2 - V_{\text{min}} / V_{\text{max}}.$$  \hspace{1cm} (4.18)$$

And it is illustrated in Figure 4.5.

### 4.3.4 Variable Stiffening: New Work

Easily noticeable is the negative curvature, or second derivative, of the graphed curve. In testing various variable stiffening methods, one may desire to create an equation for a stiffening-factor curve with positive curvature. Or perhaps one may desire a stiffening curve that produces a stiffening factor of one for the element with the maximum volume or maximum determinant of the Jacobian, thus effectively providing no stiffening to that maximum element. Since stiffening can be done however
desired, the stiffening curve can be constructed to demonstrate any feasibly desired features. In this section, I have explored creating my own stiffness curves and the resulting stiffening of the elements. All curves in this section are new stiffening curves introduced by me.

To test the effect of positive curvature on reducing distortion, I try stiffening with an elliptical stiffening-factor curve, Figure 4.6. With the elliptical curve, the hope is that I pass the first effects of distortion to the currently largest elements and effectively reduce the smallest and smaller elements to an undeformed state for each current time step. To do that, I would like to push back the point at which the curve dips down to the constraint—the ratio of the minimum to the maximum Jacobian—for the largest element or element with the maximum volume. I want to force that element to first absorb the deformation and continue sequentially from the maximum, first, down to the minimum.

So now I attempt to construct a cubic curve which allows more curvature, seen in Figure 4.7. Here, a cubic equation was formed by choosing the following constraints,
for the coefficients. In a mesh where most elements have the same volume, the ratio of $V_{\text{max}}$ to $V_{\text{min}}$ is close to 1. As the mesh becomes more distorted, the value $V_{\text{max}}/V_{\text{min}}$ increases and the value $V_{\text{min}}/V_{\text{max}}$ decreases. When the mesh requires stiffening, when there is high distortion, the ratio is very high or low. Under such assumed circumstances, the first two equations from Eq 4.19 set the stiffening factor for the element with the minimum volume very high; the element with the maximum, very low. The purpose of the following two equations is to continue greatly stiffening elements with a very small volume, even though they may not have the minimum volume. Simultaneously the elements with a very large volume, though not the largest, also need to be softened greatly. Therefore I want a small gradient near the endpoints of the curve.
Figure 4.8. Stiffness Coefficient Dependence on Element Volume.

Trying a different approach, I, again, constrain the stiffening factor to a maximum value of $V_{\text{max}}/V_{\text{min}}$ (or $J_{\text{max}}/J_{\text{min}}$ for the corresponding determinant of the Jacobian of the element) for the element with the smallest volume and a minimum value of $V_{\text{min}}/V_{\text{max}}$ for the element with the largest volume. Using those constraints, Eq 4.21 was constructed to graph a quarter portion of an ellipse between those two constraints, from the minor axis to the major:

$$\frac{(x - V_{\text{min}})^2}{(V_{\text{max}} - V_{\text{min}})^2} + \frac{(y - \frac{V_{\text{min}}}{V_{\text{max}}})^2}{(\frac{V_{\text{max}}}{V_{\text{min}}} - \frac{V_{\text{min}}}{V_{\text{max}}})^2} = 1. \tag{4.21}$$

Next, I now attempt to see if a trigonometric curve can do as well or better, shown in Figure 4.8. In this tactic, I still apply the constraints of Eq 4.19, but now to the problem

$$A \sin x + B \sin x^2 + C \sin x^3 + D \sin x^4 = 0, \tag{4.22}$$

where the coefficients represent the unknowns.
The results are very similar between the different cases. Illustrated in Table 4.6, one can see that there is not much difference between the different variable methods. This suggests that no single curve was optimal. If for a given mesh at a given time step, one can pinpoint the exact optimal point for the stiffening-factor curve to dip down toward the maximum volume or maximum determinant of the Jacobian, it may be possible to pleasantly and substantially reduce distortion. Of course this optimal curve would vary with every time step. And so it creates a formidable problem to determine at each step. A measurement would have to be made as to the remeshing and projection cost savings versus the costs of this incremental determination.

4.3.5 Strain–Based Stiffening: Previous Work

The stiffness can also be varied based on the strain so that elements undergoing more strain receive more relative stiffening. Proposed by Chianduissi, Bugeda, and Oñate [31], the element strain is derived from the stress using a uniform stiffness (Equation 4.3–4.5). Then with this predicted element strain, I relatively stiffen the equation with an appropriate stiffening coefficient.

4.3.6 Strain–Based Stiffening: New Work

I have taken Chianduissi’s idea and applied it in 3D to the problem of a paratrooper jumping from a cargo aircraft. This new application constitutes a new work. The purpose here is to determine if this tactic can be used to stiffen and facilitate the paratrooper–aircraft separation problem, allowing its computation with fewer meshes.

From Table 4.7, the results from this method are not especially better than any of the other stiffening results. The results are average. And this leads to the conclusion that the stress, and therefore the strain, is not enough to predict which element has the highest probability of reaching the distortion limit in the next time step. If this factor is found, I believe the stiffening will have much better results than any of the previous methods.
4.3.7 Material Property Stiffening: Previous Work

This method is helpful in situations where different forcing functions are tried. Depending on the force the material undergoes, certain materials will obviously deform more readily and much faster than other materials. And that is exactly what this method achieves: it changes the material constant of the mesh material thereby creating a stiffer or a more flexible substance or pseudo-solid. But this method does not change the stiffness of any element in relation to the other elements.

4.3.8 Material Property Stiffening: New Work

Here, again, I apply this stiffening tactic to my 3D problem of a paratrooper separating from a cargo aircraft. This is done to test its effectiveness within the context of 3D simulations and specifically with my application. The results are the same.

As in Table 4.8, Johnson has already shown [18] that given a deformation of the domain and holding all other mesh stiffening constant, element distortion increases with an increasing value for $\lambda/\mu$ (Equation 4.4). And I found this to be true. This subsection illuminates the importance of the optimal choice for the material properties to minimize distortion. As Johnson showed, lower is better; therefore, a value of one is usually taken. The simulation for the value of 0.001 required so much time, it was discontinued.

4.3.9 Aspect Ratio–Based Stiffening: New Contribution

This successful mesh stiffening tactic, newly introduced here, has proven to result in the most time steps between remeshes, compared to the aforementioned methods. It has both the highest observed number of time steps for any one mesh and the highest average number of time steps per mesh (Table 4.9).

According to this method, I stiffen the elements based on the aspect ratio. Elements with higher aspect ratios receive a higher stiffening coefficient for their mesh
moving equation. As a relational experiment, I also attempted to stiffen the elements according to their volume ratio, which is defined by \( \log \frac{V}{V_0} \), where \( V_0 \) is the original undeformed volume of the element. According to Table 4.9, stiffening by the volume ratio did not perform as well as stiffening by the aspect ratio. And this is understandable since an element can undergo a large increase in volume ratio as the volume changes, yet still retain the same aspect ratio: imagine an element that is stretched equally in all axially directions. With this possibility, elements with fair aspect ratios can be stiffened more than elements with very high aspect ratios and poor volume ratios. In essence, stiffening according to the volume ratio, tracks general deformation, whereas aspect ratio–based stiffening tracks distortion which is what is of most importance to prevent the loss of accuracy in the solution.

I believe this is due to the type of attack on distortion that the aspect ratio–based stiffening employs. The root of the distortion problem is mesh moving. The movement of the mesh distorts the elements. I directly attack this root problem through mesh moving alternatives in Chapter 5, thereby eliminating distortion. However, if the desire to indirectly minimize distortion still exists while ignoring the root of the problem and allowing mesh moving, the most effective route is to attack the symptom, itself—distortion. Since distortion is measured by the aspect ratio (Equations 4.10–4.12), then the best indirect method is one that increasingly stiffens elements with higher aspect ratios. And that is why this new tactic produces the best results. In other words, for the same problem, it is the method that allows for the most time steps within the same mesh, or before a remesh is required.

### 4.4 Traditional Remeshing

Regardless of the mesh stiffening method employed, after some time the mesh becomes distorted beyond an acceptable limit, and remeshing is required. This is required to restore accuracy again to the solution by generating new elements with aspect ratios as close to a value of 1.0 as possible. That is very different than the aspect ratios of the elements in the current distorted mesh.
In remeshing during a mesh moving Fluid–Object Interaction simulation, I first extract an inner surface mesh of the currently positioned particle, or object. If there are multiple particles, one surface mesh file is created containing the newly located surface meshes of all particles, or objects. Next, I combine the surface mesh of the particle(s) with a surface mesh of the entire domain. I then, using an automatic mesh generator, generate a new 3D mesh with new well–proportioned elements and the new location of the particle(s).

However, in Section 4.2.4, the idea of moving and nonmoving zones was introduced. Using such a zonal mesh requires a modification in the remeshing procedure, and since I normally use zones in mesh moving simulations, this modification is very common. Instead of having a prepared surface mesh of the entire domain ready to combine with the new surface mesh of the particle(s), a surface mesh is prepared only for the moving zone. Since the elements in the nonmoving zone are never distorted because they never move, that zone does not need to be remeshed. In this case, I have a prepared 3D mesh of the nonmoving zone set aside.

From the final 3D mesh when the simulation has stopped, I, again, extract a surface mesh of the newly positioned particle(s). Secondly, I combine this surface mesh with a surface mesh of the moving zone, only. Thirdly, using an automatic mesh generator, I produce a well–proportioned 3D mesh of the moving zone. Lastly, I combine this moving zone 3D mesh with the prepared 3D mesh of the nonmoving zone, thus, giving me a new 3D mesh of the newly–located particle and the entire domain containing elements with aspect ratios equal to or very near 1.0.

4.5 Projection

Once a new mesh has been generated, during a mesh moving simulation, the solution from the old mesh must be projected onto the new mesh before the simulation can continue. The projection is necessary because the nodes of the new mesh are not in the same location as those in the old mesh. So the values of velocity and pressure must be determined at the new locations. There are many different methods of
projection, and I will briefly review them as Johnson originally surveyed them, [18].

4.5.1 Conventional Projection

Linear Interpolation This is the simplest, easiest, and most highly used method. Here, using linear shape functions, I map the physical space coordinates in the new mesh to those in the old mesh and perform a search in the old mesh to find the old–mesh element that contains the new node using the equations

$$X(r, s, t) = \sum_{i=1}^{n_{en}} N_i(r, s, t) X_i^*,$$  \hspace{1cm} (4.23)

$$d(r, s, t) = \sum_{i=1}^{n_{en}} N_i(r, s, t) d_i^*.$$  \hspace{1cm} (4.24)

Here, the physical space coordinate is represented by $X$ while the element local space is represented by $(r, s, t)$. Terms related to the old mesh are denoted by a *. Once the new node location is found in the old–mesh element, Equation 4.23 can then be inversely solved to find the local coordinates in the old–mesh, $(r, s, t)$. Now, using another equation, Equation 4.24, I can solve for the value of the unknown at this new node position.

Second--Order Interpolation In this type of interpolation, not much is varied, but the order of the interpolation. Here, the $d_i^*$, in Equation 4.24, is expanded to include first and second order derivatives. This is done using a Taylor Series expansion of the unknown in the old mesh, [32].

Least Squares Interpolation In this scheme, a finite element formulation is utilized to solve for the unknown in the new mesh. The goal is to minimize the square of the absolute value of the difference between the unknown in the new mesh and that of the old mesh,

$$(\|d - d^*\|)^2,$$  \hspace{1cm} (4.25)
where \( \mathbf{d} \) is the unknown in the new mesh and \( \mathbf{d}^* \) is that of the old mesh. In minimizing this function, I set the derivative to zero. Dividing out the constant and integrating the equation over the entire domain gives

\[
\int_{\Omega} \mathbf{w} \cdot (\mathbf{d} - \mathbf{d}^*) \, d\Omega = 0,
\]

where \( \mathbf{w} \) is the variation of \( \mathbf{d} \) in the new mesh, \( \mathbf{d}^* \) being known. Even though \( \mathbf{d} \) and \( \mathbf{d}^* \) are in two different meshes, one mesh must be chosen over which to do the integration. I choose the location of the unknown \( \mathbf{d} \)—the new mesh. Since \( \mathbf{d}^* \) is known, it is more convenient to find its values in the new mesh than to find the values of the unknown in the old mesh. Therefore I must find the values of \( \mathbf{d}^* \) in the new mesh in order to carry out the integration. And this is done at the Gaussian integration points of the elements in the new mesh. Finding the values of the integration points in physical space, Equations 4.23 and 4.24 can be used to find those values in the new mesh.

Once the equation is formulated, it is solved iteratively, or incrementally. The kth iteration uses the equation

\[
\mathbf{d} = \mathbf{d}_k + \Delta \mathbf{d}_k.
\]  

Substituting for \( \mathbf{d} \) back in Equation 4.26, I achieve

\[
\int_{\Omega} \mathbf{w} \cdot \Delta \mathbf{d}_k = \int_{\Omega} (\mathbf{d}^* - \mathbf{d}_k) \, d\Omega = 0,
\]

which, in matrix notation, takes on the form of

\[
\mathbf{M} \Delta \mathbf{d}_k = \mathbf{F}_k.
\]  

(4.29)
where $\mathbf{M}$ is the mass matrix, $\Delta \mathbf{d}_k$ is the vector of nodal unknowns and $\mathbf{F}_k$ is a force vector. In my computations, I approximate $\mathbf{M}$ with the lumped mass matrix, $\mathbf{M}_L$ simplifying the calculations and allowing for ease in inverting the matrix,

$$
\Delta \mathbf{d}_k = (\mathbf{M}_L)^{-1} \mathbf{F}_k.
$$

(4.30)

This iterative process is carried out until the residual is reduced to a predetermined acceptable limit. This process represents a slight variation of the Jacobi iterative technique where the diagonal of $\mathbf{M}$ is used, instead.

Now, using the Least Squares Interpolation technique, there are two subtypes of projection interpolation schemes introduced by Johnson, [18]. They are subtypes because they are used in conjunction with the Least Squares Projection technique. These schemes are concerned with how the new projected values are integrated, again, into the incompressible flow calculations, as the computations continue.

**Jump Term Interpolation** This is the simplest scheme requiring the least work in the space–time formulation. The projected values for velocity and pressure are used as the upper level values of velocity and pressure for the previous time slab. So when continuing with the incompressible flow formulation, Equation 3.12, the values for the vector $(\mathbf{u}^h)^-\nu$, in the jump term, are acquired from the projected solution.

Johnson points out that because this equation is similar in structure to Equation 4.26, then the Least Squares Interpolation can be incorporated into the general finite element formulation of the fluid flow problem. And this is done through the jump term interpolation scheme. The projected solution is used as the initial guess for the unknowns in the fluid flow problem.

**Pressure Adjusted Interpolation** The problem with the jump term interpolation scheme is that the projected solution does not satisfy the incompressibility constraint. Whenever a projection occurs, a slight violation in this constraint arises.
This is most likely due to the approximation from the numerical integration of the jump term using two meshes. This slight violation causes a spike or jump in the pressure forces acting on objects in moving simulations, or fluid object interactions. The pressure, acting as a Lagrange multiplier, reacts to the violation of the incompressibility constraint which is incorporated into Equation 3.12.

Because of the slight violation in the incompressibility constraint and the fact that the pressure, to be calculated in the first time step after remeshing, will be incorrectly spiked, either up or down, I choose not to use this calculated pressure. The pressure variable is ignored, and the interpolated pressure from the Least Squares Interpolation, is used again for that first time step. In the aforementioned jump term interpolation scheme, the pressure variable is calculated during the first time step. Contrastingly, here, in the pressure adjusted scheme, the pressure variable containing the spiked pressure is ignored at the end of the first time step, and the interpolated pressure is used, again, as the “calculated” pressure for the first time step. However, the results have never been conclusive.

4.5.2 Compressibly Adjusted Interpolation: A New Tactic

The following scheme is a new tactic, introduced here to reduce pressure spikes due to the violation of the incompressibility constraint. Unlike the inability of the pressure adjusted interpolation scheme to firmly solve the problem [18], this scheme does conclusively reduces those spikes. It also allows variability in the amount of damping applied to the pressure spikes.

As stated, in many simulations, the results of the Pressure Adjusted Interpolation scheme can be inconclusive. In other words, the scheme does not reduce the pressure spikes. They still exist or are barely mollified. When this occurs, it must be concluded the solution has not yet reached a divergence-free state. To restate, in the Pressure Adjusted Interpolation scheme, for the first time step, when the projection pressure is used instead of the calculated pressure, if the pressure still spikes subsequently, I
believe the velocity field is still not divergence-free. If it were, the pressure would no longer retain the tendency to jump or spike. The ineffectiveness of that scheme in some simulations led to my new scheme: the Compressibly Adjusted Interpolation scheme. First, I realize that to use the projected pressure after the first non-linear iteration after a remesh, and to continue on with the non-linear iterations simply reinforces the incompressibility constraint. Since the field is not divergence-free, to continue on with the non-linear iterations forces the pressure to react and jump in an effort to enforce the incompressibility restraint. So the first tenant of this scheme is that whenever reusing the projected pressure, or clipping the pressure, I should stop the non-linear iterations and immediately move on to the next time step. This is a way of allowing compressibility into the formulation until the velocity field becomes divergence-free. The second tenant of this scheme, is that as long as the velocity-field is still not divergence-free, the pressure can still be clipped—using the projected pressure instead of the newly calculated pressure. This is to say that the pressure can be clipped for more than one time step. Looking at Figures 4.9-4.13 you can see the effect of clipping the pressure in this scheme for one, two, and four time steps.

It is important to note that the simulations that clip the longest are closest to retaining the pre-remesh trajectory after remeshing. Secondly, clipping for two time steps reduces the pressure jump to a lower level than that when clipping for one time step; clipping for four time steps has a smaller jump than when clipping for two time steps. Each figure shows a simulation that starts after a remesh, computes, stops, remeshes again halfway through the computation, and computes, again, through to the next stop due to distortion. All figures showing simulations with multiple clips, move immediately to the next time step after clipping.

With pressure clipping, the pressure spikes are reduced conclusively. Moreover, without the pronounced pressure spikes, the general trajectory of the force felt by the paratrooper before a remesh is maintained after a remesh. This is maintained better than it was with no pressure clipping.
4.5.3 A New Projection Method: Divergence–Free Projection

There is considerable work invested into the effort to reduce the error effects of remeshing and projection when a remesh is required. If there were a way not just to reduce but, even more, to eliminate those errors, it would be greatly needed and utilized. And so, my contribution—a Divergence–Free Projection method—is introduced here.

It is important to note, however, that pressure spikes, or jumps, have two sources: the numerical integration over a new mesh with different nodal coordinates (remeshing) and increased accuracy from a better quality remesh. In other words, if I remeshed without changing the connectivity, there would still be a pressure jump due to the increase in accuracy from a more egalitarian dispersion of nodes throughout the mesh. Before a remesh, in a distorted mesh, there are areas of high concentration of nodes and areas of low concentration, decreasing the accuracy of the solution. This is corrected by remeshing. But this new equal distribution of nodes will display
different force levels on an object.

Divergence–Free Projection, my new method, becomes an attempt to correct pressure jumps caused by the former source of pressure spikes: the numerical integration which disturbs the divergence–free condition. So the constraint of incompressibility on the projection equations is employed, here, to correct for that. And desirously, the reaction of the pressure to divergence in the first time steps after a remesh will be eradicated.

Decomposing Equation 4.25 into its scalar components, the projection equation system is constrained by a divergence–free condition producing the goals to

\[
\text{Minimize}(\|u - u^*\|^2), \\
\text{Minimize}(\|v - v^*\|^2), \\
\text{Minimize}(\|w - w^*\|^2), \\
\text{Minimize}(\|p - p^*\|^2),
\]
where the velocity vector $\mathbf{u} = (u, v, w)$, $p$ represents pressure. Setting to zero (followed by taking the derivative and dividing out the coefficient for the first four
equations) and integrating over the domain $\Omega$ is completed to produce

\[
\int_{\Omega} w_1 (u - u^*) \, d\Omega = 0, \\
\int_{\Omega} w_2 (v - v^*) \, d\Omega = 0, \\
\int_{\Omega} w_3 (w - w^*) \, d\Omega = 0, \\
\int_{\Omega} w_4 (p - p^*) \, d\Omega = 0, \\
\int_{\Omega} q (\nabla \cdot \mathbf{u}) \, d\Omega = 0, 
\]  

(4.32)

where $q$ and the four $w$ variables are scalar weighting functions within the finite element formulation as the square of the residual is minimized.

In solving this equation system, different techniques than those used for the solution of the Navier–Stokes equations, were employed. These fall in three areas: the iterative solver, the preconditioner, and the matrix storage method.

Before constraining with the divergence–free condition, the equation system was square. Now, by adding this equation, the number of equations was increased by
one while the number of unknowns remained the same. With four unknowns and five equations per nodal block in the matrix, the system is overdetermined. For this reason, instead of using the GMRES iterative solver, [33], that is used in my flow solvers for the Navier–Stokes equations of incompressible flows, the Conjugate Gradient for the normal equations (CGNE), or Craig’s method, was chosen for this projection solver [34].

1. Compute $r_0 = b - Ax_0, p_0 = A^T r_0$.

2. For $i = 0, 1, \ldots$, until convergence Do:

3. $\alpha_i = (r_i, r_i)/(p_i, p_i)$

4. $x_{i+1} = x_i + \alpha_i p_i$

5. $r_{i+1} = r_i - \alpha_i A p_i$

6. $\beta_i = (r_{i+1}, r_{i+1})/(r_i, r_i)$

7. $p_{i+1} = A^T r_{i+1} + \beta_i p_i$

8. EndDo

Algorithm 4.1: CGNE

In Algorithm 4.1, $r_i$ and $x_i$ are the residual and solution vectors during the $i$th iteration, respectively. The matrix $A$ is used to produce the conjugate direction $p_i$ which is used during the $i$th iteration. The values $\alpha_i$ and $\beta_i$ are intermediate values during the $i$th iteration determining the degree each conjugate direction is searched.

If CGNE were enough for my problem, I would stop, but the matrix is not well-conditioned enough for convergence with CGNE. So a preconditioner must be utilized to aid convergence towards a solution. Following Saad’s work [15], the Incomplete LQ preconditioner (ILQ) was chosen; it displays the best convergence for solving normal equations, and there is guarantee of the existence of an ILQ factorization. According to Saad, the procedure follows as such.
1. For $i = 1, 2, \ldots, N$ Do:

2. Step 1. If $i = 1$ define $\mathbf{q}_1 = \mathbf{a}_1$ and goto Step 5.

3. Step 2. Compute all nonzero inner products $l_{ij} = \mathbf{q}_j \mathbf{q}_i^T, j = 1, 2, \ldots, i - 1$.

4. Step 3. Determine the $p_L$ largest $l_{ij}$'s $j = 1, 2, \ldots, i - 1$ and assign a zero value to the others.

5. Step 4. Compute $\mathbf{q}_i = \mathbf{a}_i - \sum_{j=1, l_{ij} \neq 0}^{j=i-1} l_{ij} \mathbf{q}_j$

6. Step 5. Determine the $p_Q$ largest elements of $\mathbf{q}_i$ and assign a zero value to the others.

7. Step 6. Compute $l_{ii} = (\|\mathbf{q}_i\|)_2$ and $\mathbf{q}_i = \mathbf{q}_i / l_{ii}$

Algorithm 4.2: Incomplete LQ Factorization

In Algorithm 4.2, $\mathbf{a}_i$ and $\mathbf{q}_i$ represent the $i$th row of matrices $\mathbf{A}$ and $\mathbf{Q}$, respectively, while $l_{ij}$ are inner products. The tolerances $p_L$ and $p_Q$ are left to the user to determine, but can be limited by the maximum memory capacity for very large matrices. With more memory one can have a higher tolerance. This leads to issues of memory storage techniques. The matrix $\mathbf{A}$ for the fluid–object interaction and aircraft separation problems are large, and proper storage techniques should be utilized to minimize the load on the computer memory.

The newly introduced matrix storage technique used for $\mathbf{A}$ in this projection scheme is a variation of a Block Sparse Row (BSR) Format. In BSR, there are three arrays: $\mathbf{AA}$, holding the non–zero values of $\mathbf{A}$; $\mathbf{IA}$, containing the block row index of each non–zero block; and $\mathbf{JA}$, containing the block column index of each non–zero. In the Block Sparse Row format, only the non–zero elements of the non–zero blocks are stored, and the structure of the non–zero elements in each block is known. With that knowledge, the stored elements in $\mathbf{AA}$ can be arranged to place each value in its correct position, which becomes useful in matrix operations.
Here, in this work, the new matrix storage technique is called Non-Repeating Block Sparse Row (NRBSR) Format. In NRBSR, I not only discard the zero elements of the non–zero blocks, but also the formulaically repetitious values. In the non-square projection matrix, the values on the diagonal of the non-square blocks are all the exact same, formulaically. I store the value only once discarding three extra repetitious values per block. Figure 4.14 shows the block fill–in for Equations 4.32.

Finally, the matrices L and Q are stored using the normal Compressed Row Storage (CSR) technique. Matrix Q is discarded once the ILQ procedure is completed because its orthogonality allows it to fall out of the equation. Algorithm 4.3 shows the CGNE algorithm as employed in this projection scheme, with split preconditioning.

1. Compute \( r_0 = b - Ax_0, \hat{r}_0 = L^{-1}r_0p_0 = A^T L^{-T} r_0 \).

2. For \( j = 0, 1, \ldots \), until convergence Do:
3. \[ \theta_j = Ap_j \]

4. \[ \alpha_i = (r_j, r_j)/(p_j, p_j) \]

5. \[ x_{j+1} = x_j + \alpha_j p_j \]

6. \[ r_{j+1} = r_j - \alpha_j L^{-1} w_i \]

7. \[ \beta_j = (r_{j+1}, r_{j+1})/(r_j, r_j) \]

8. \[ p_{j+1} = A^T L^{-T} r_{j+1} + \beta_j p_j \]

9. EndDo

Algorithm 4.3: CGNE with Split Preconditioning

In Algorithm 4.3, \( \hat{r} \) is the residual of the preconditioned system, and \( L^{-1} \) and \( L^{-T} \), the inverse of the transpose of \( L \), are the preconditioners. The vector \( w \) has been added for readability and ease.

The CGNE algorithm could have been used with left, right, split, or centered preconditioning. Left and right preconditioned algorithms have identical iterates if the proper inner products are used for either. So the choice was between three: left/right, split, or centered. In centered preconditioning a different preconditioner is needed, one that makes \( AM^{-1}A^T \approx I \). And so the \( L \) from the ILQ method can only be used with left/right or split preconditioning. Split preconditioning was then chosen to preserve symmetry.

Figure 4.15 shows the results of the new divergence-free projection solver on a simulation of a paratrooper falling from a cargo aircraft, both before and after a particular remesh. The new divergence-free projection solver removed the part of the pressure spike related to the inaccurate numerical integration across two meshes with different elements and nodal coordinates. However, the parts of the pressure jumps due to a departure from the degradation of the solution through distorted elements was not eliminated and should not be eradicated. This part of the jump (and the part due to different locations of the nodes on the body in question, which should
reduce with increased refinement) is good; a more accurate solution is desired and is the reason I remesh—to improve the solution. If the solution degrades so badly that large pressure spikes result from only this part, the redistribution of elements and loss of distortion, then remeshes should be scheduled more frequently or much more earlier. A lower distortion limit should be set so that this jump is small. Alluded to earlier, the third part due to the different location of the nodes goes away with increased refinement. A very high refinement reduces pressure jumps when remeshing because with higher refinement, each node on the body carries or experiences a smaller amount of the integrated force. So then, true to its intention, the newly introduced divergence-free solver was successful in eliminating spikes or the parts of pressure spikes due to previous projection error. A comparison to the pressure clipping methods is shown in Figure 4.16. It outperforms pressure clipping which only reduces the spikes.
Figure 4.16. Non-linear Time Steps Before and After Remeshing for a Paratrooper Falling from a Cargo Aircraft.
Table 4.1. First Paratrooper Simulation: Time Steps per Mesh.

<table>
<thead>
<tr>
<th>Meshe$\chi$</th>
<th>$\chi = 0$</th>
<th>$\chi = 1$</th>
<th>$\chi = 2$</th>
<th>$\chi = 3$</th>
<th>$\chi = 4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>79</td>
<td>79</td>
<td>79</td>
<td>79</td>
<td>79</td>
</tr>
<tr>
<td>2</td>
<td>93</td>
<td>93</td>
<td>91</td>
<td>76</td>
<td>83</td>
</tr>
<tr>
<td>3</td>
<td>73</td>
<td>67</td>
<td>75</td>
<td>64</td>
<td>71</td>
</tr>
<tr>
<td>4</td>
<td>53</td>
<td>52</td>
<td>60</td>
<td>59</td>
<td>59</td>
</tr>
<tr>
<td>5</td>
<td>53</td>
<td>50</td>
<td>51</td>
<td>59</td>
<td>52</td>
</tr>
<tr>
<td>6</td>
<td>44</td>
<td>52</td>
<td>46</td>
<td>53</td>
<td>59</td>
</tr>
<tr>
<td>7</td>
<td>58</td>
<td>48</td>
<td>47</td>
<td>50</td>
<td>48</td>
</tr>
<tr>
<td>8</td>
<td>37</td>
<td>41</td>
<td>45</td>
<td>0</td>
<td>45</td>
</tr>
<tr>
<td>9</td>
<td>44</td>
<td>42</td>
<td>41</td>
<td>44</td>
<td>43</td>
</tr>
<tr>
<td>10</td>
<td>38</td>
<td>43</td>
<td>38</td>
<td>41</td>
<td>39</td>
</tr>
<tr>
<td>11</td>
<td>31</td>
<td>37</td>
<td>33</td>
<td>39</td>
<td>38</td>
</tr>
<tr>
<td>12</td>
<td>32</td>
<td>36</td>
<td>38</td>
<td>40</td>
<td>36</td>
</tr>
<tr>
<td>13</td>
<td>33</td>
<td>35</td>
<td>35</td>
<td>36</td>
<td>34</td>
</tr>
<tr>
<td>14</td>
<td>37</td>
<td>32</td>
<td>32</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>15</td>
<td>31</td>
<td>27</td>
<td>29</td>
<td>31</td>
<td>33</td>
</tr>
<tr>
<td>16</td>
<td>31</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
</tr>
<tr>
<td>17</td>
<td>34</td>
<td>27</td>
<td>28</td>
<td>33</td>
<td>26</td>
</tr>
<tr>
<td>18</td>
<td>37</td>
<td>25</td>
<td>30</td>
<td>33</td>
<td>27</td>
</tr>
<tr>
<td>19</td>
<td>33</td>
<td>30</td>
<td>25</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>20</td>
<td>35</td>
<td>27</td>
<td>27</td>
<td>26</td>
<td>25</td>
</tr>
<tr>
<td>21</td>
<td>39</td>
<td>28</td>
<td>33</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>30</td>
<td>26</td>
<td>25</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>23</td>
<td>27</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td>23</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>26</td>
<td>31</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>33</td>
<td>37</td>
<td></td>
<td></td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>$\chi = 0$</td>
<td>$\chi = 1$</td>
<td>$\chi = 2$</td>
<td>$\chi = 3$</td>
<td>$\chi = 4$</td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td><strong>Meshes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>28</td>
<td>42</td>
<td></td>
<td>37</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>38</td>
<td>18</td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td>17</td>
<td></td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td><strong>Total Steps</strong></td>
<td>1026</td>
<td>1098</td>
<td>1096</td>
<td>934</td>
<td>1102</td>
</tr>
<tr>
<td><strong>Maximum</strong></td>
<td>93</td>
<td>93</td>
<td>91</td>
<td>79</td>
<td>83</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>42.75</td>
<td>39.21</td>
<td>40.59</td>
<td>40.61</td>
<td>38</td>
</tr>
</tbody>
</table>
Table 4.3. Second Paratrooper Simulation: Time Steps per Mesh.

<table>
<thead>
<tr>
<th>Meshes</th>
<th>$\chi = 1$</th>
<th>$\chi = 2$</th>
<th>$\chi = 3$</th>
<th>$\chi = 5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>36</td>
<td>23</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>27</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>96</td>
<td>67</td>
<td>43</td>
<td>44</td>
</tr>
<tr>
<td>4</td>
<td>73</td>
<td>55</td>
<td>39</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>62</td>
<td>51</td>
<td>34</td>
<td>44</td>
</tr>
<tr>
<td>6</td>
<td>55</td>
<td>45</td>
<td>34</td>
<td>32</td>
</tr>
<tr>
<td>7</td>
<td>46</td>
<td>37</td>
<td>38</td>
<td>18</td>
</tr>
<tr>
<td>8</td>
<td>48</td>
<td>47</td>
<td>24</td>
<td>25</td>
</tr>
<tr>
<td>9</td>
<td>16</td>
<td>38</td>
<td>29</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>39</td>
<td>35</td>
<td>26</td>
<td>20</td>
</tr>
<tr>
<td>11</td>
<td>40</td>
<td>35</td>
<td>19</td>
<td>29</td>
</tr>
<tr>
<td>12</td>
<td>41</td>
<td>32</td>
<td>24</td>
<td>21</td>
</tr>
<tr>
<td>13</td>
<td>34</td>
<td>30</td>
<td>22</td>
<td>19</td>
</tr>
<tr>
<td>14</td>
<td>34</td>
<td>33</td>
<td>23</td>
<td>20</td>
</tr>
<tr>
<td>15</td>
<td>36</td>
<td>27</td>
<td>22</td>
<td>25</td>
</tr>
<tr>
<td>16</td>
<td>32</td>
<td>31</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>17</td>
<td>30</td>
<td>26</td>
<td>21</td>
<td>10</td>
</tr>
<tr>
<td>18</td>
<td>25</td>
<td>26</td>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>19</td>
<td>30</td>
<td>23</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>27</td>
<td>25</td>
<td>13</td>
<td>3</td>
</tr>
<tr>
<td>21</td>
<td>29</td>
<td>23</td>
<td>17</td>
<td>10</td>
</tr>
<tr>
<td>22</td>
<td>34</td>
<td>21</td>
<td>17</td>
<td>40</td>
</tr>
<tr>
<td>23</td>
<td>29</td>
<td>20</td>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td>24</td>
<td>26</td>
<td>20</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>3</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>Meshes</td>
<td>( \chi = 1 )</td>
<td>( \chi = 2 )</td>
<td>( \chi = 3 )</td>
<td>( \chi = 5 )</td>
</tr>
<tr>
<td>--------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>26</td>
<td>28</td>
<td>23</td>
<td>19</td>
<td>7</td>
</tr>
<tr>
<td>27</td>
<td>29</td>
<td>21</td>
<td>14</td>
<td>9</td>
</tr>
<tr>
<td>28</td>
<td>56</td>
<td>22</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>29</td>
<td>17</td>
<td>24</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>30</td>
<td>5</td>
<td>20</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>31</td>
<td>23</td>
<td>15</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>19</td>
<td>13</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>21</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>22</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>16</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>21</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>22</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>19</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>20</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>22</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>18</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>13</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>3</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Table 4.5. Second Paratrooper Simulation Time Steps Per Mesh—Continued.

<table>
<thead>
<tr>
<th></th>
<th>$\chi = 1$</th>
<th>$\chi = 2$</th>
<th>$\chi = 3$</th>
<th>$\chi = 5$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Meshes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
<td></td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>Total Steps</strong></td>
<td>1106</td>
<td>1127</td>
<td>698</td>
<td>747</td>
</tr>
<tr>
<td><strong>Maximum</strong></td>
<td>96</td>
<td>67</td>
<td>43</td>
<td>13.58</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>36.87</td>
<td>25.61</td>
<td>21.81</td>
<td>44</td>
</tr>
</tbody>
</table>

Table 4.6. Paratrooper Simulation: Variable Stiffening.

<table>
<thead>
<tr>
<th></th>
<th>Regular</th>
<th>Variable</th>
<th>Ellipse</th>
<th>Cubic</th>
<th>Sine</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Meshes</strong></td>
<td>30</td>
<td>31</td>
<td>61</td>
<td>70</td>
<td>54</td>
</tr>
<tr>
<td><strong>Total Steps</strong></td>
<td>1106</td>
<td>1104</td>
<td>1100</td>
<td>1108</td>
<td>1109</td>
</tr>
<tr>
<td><strong>Maximum</strong></td>
<td>96</td>
<td>82</td>
<td>44</td>
<td>62</td>
<td>70</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>36.87</td>
<td>35.61</td>
<td>18.33</td>
<td>15.82</td>
<td>20.54</td>
</tr>
</tbody>
</table>

Table 4.7. Paratrooper Simulation: Strain-Based Stiffening.

<table>
<thead>
<tr>
<th></th>
<th>Regular</th>
<th>$1$–Norm</th>
<th>$2$–Norm</th>
<th>$F$–norm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Meshes</strong></td>
<td>30</td>
<td>38</td>
<td>33</td>
<td>40</td>
</tr>
<tr>
<td><strong>Total Steps</strong></td>
<td>1106</td>
<td>1072</td>
<td>1130</td>
<td>1107</td>
</tr>
<tr>
<td><strong>Maximum</strong></td>
<td>96</td>
<td>72</td>
<td>93</td>
<td>70</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>36.87</td>
<td>28.21</td>
<td>34.24</td>
<td>27.68</td>
</tr>
</tbody>
</table>
Table 4.8. Paratrooper Simulation: Material Stiffening.

<table>
<thead>
<tr>
<th></th>
<th>1.0</th>
<th>10</th>
<th>100</th>
<th>1000</th>
<th>0.1</th>
<th>0.01</th>
<th>0.001</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Meshes</strong></td>
<td>30</td>
<td>57</td>
<td>88</td>
<td>96</td>
<td>lll</td>
<td>256</td>
<td>—</td>
</tr>
<tr>
<td><strong>Total Steps</strong></td>
<td>1106</td>
<td>1103</td>
<td>1031</td>
<td>1035</td>
<td>1265</td>
<td>1556</td>
<td>—</td>
</tr>
<tr>
<td><strong>Maximum</strong></td>
<td>96</td>
<td>44</td>
<td>25</td>
<td>22</td>
<td>12</td>
<td>9</td>
<td>—</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>36.87</td>
<td>19.35</td>
<td>11.72</td>
<td>10.78</td>
<td>11.40</td>
<td>6.08</td>
<td>—</td>
</tr>
</tbody>
</table>

Table 4.9. Paratrooper Simulation: Aspect Ratio–Based Stiffening.

<table>
<thead>
<tr>
<th></th>
<th>Regular</th>
<th>Volume1</th>
<th>Volume2</th>
<th>Aspect1</th>
<th>Aspect2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Meshes</strong></td>
<td>30</td>
<td>58</td>
<td>89</td>
<td>25</td>
<td>20</td>
</tr>
<tr>
<td><strong>Total Steps</strong></td>
<td>1106</td>
<td>1091</td>
<td>1121</td>
<td>969</td>
<td>889</td>
</tr>
<tr>
<td><strong>Maximum</strong></td>
<td>96</td>
<td>49</td>
<td>33</td>
<td>142</td>
<td>220</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>30.87</td>
<td>18.81</td>
<td>12.60</td>
<td>38.76</td>
<td>44.45</td>
</tr>
</tbody>
</table>
Chapter 5

Fluid–Object Interactions
Subcomputation Technique

Heretofore, when the costs of mesh moving, remeshing, and projection have become to great or cumbersome, the alternative has been a point mass subcomputation. First, a flow would be modeled in a main computation without the special object in question (in these problems, a separating object). Then the main flow would be used to determine the boundary conditions for the subcomputational mesh only containing the point mass. The subcomputation would predict some translational movement which would lead to the new location in the main computation from which to derive boundary conditions for the next time step. And the steps are repeated, again.

In this chapter, a new technique is presented as a better approximating alternative to mesh moving than the point mass method: Fluid–Object Interactions Subcomputation Technique (FOIST). Many different approaches have been taken in fluid–structure coupling with numerous variations. In Section 5.1, the methodology is explicated. And some of the key issues involved in the approach are presented in 5.2. FOIST is demonstrated through test cases in Section 5.3. This chapter will contain no delineation between previous work and new contributions as the entire chapter is a new contribution.
5.1 New FOIST Method

FOIST is an intermediate approximation between mesh moving and a point mass subcomputation. In order for FOIST to be effective, it must be possible to make one assumption: the effect of the special objects or smaller-scale objects on the main flow field is negligible. It is still true that the main flow field effects the smaller objects, but in FOIST I only have this direction of influence. This allows me to create a one-way coupling. In other words, the main flow field is computed without the presence of the smaller separating objects. So the dynamics of one of these objects are determined by carrying out a subcomputation around that object. The boundary conditions for the subcomputation come directly from the main flow field. Once the subcomputation is carried out, the resulting incremental translation and rotation are determined and used to find the new and next location in the main flow field that the object would have moved to if mesh moving were performed. After the new location is found, the boundary conditions, once again, are projected from the main flow field to the boundaries of the subcomputation, and the cycle is repeated.

So the main flow field is computed using a fixed mesh since there is no moving or separating object in the domain. Likewise, the subcomputation flow field is also computed using a fixed mesh. I simply update the boundary conditions for the subcomputation at each instant, and as stated earlier, use the moments and forces to calculate the translation and rotation to find the next location and orientation within the main flow field.

As FOIST continues, each translational and rotational increment is stored. This allows me to compute a path or trajectory for the object, the goal of my mesh moving alternative. If FOIST can be used to find a good approximation of a trajectory of a separating object without using mesh moving, then it has done its job well, since the costs were reduced. And it is the trajectory of some test cases that will be analyzed to determine the efficacy of FOIST.
5.2 FOIST Issues

There are a few issues that arise while performing FOIST for any simulation. Most are all related to the boundary conditions of the subcomputation. Issues for future study and direction can be found in Chapter 9.

5.2.1 FOIST Trajectory Starting Point

In many separation problems, the smaller object separating from the larger object starts very close to the larger object. In the case of a paratrooper jumping from an AirForce cargo aircraft, the paratrooper even starts inside the aircraft! Because of this, FOIST cannot always begin a trajectory computation at the same location mesh moving simulations can commence. This is due to the need to derive boundary conditions for the subcomputation. So the rectangularly prismatic domain for the subcomputation was fit fully in the fluid mesh between the aircraft in the main computational domain and the walls of the domain. In this example, to avoid hitting the sides of the aircraft with the invisible subcomputational domain walls, the FOIST subcomputation must begin with the paratrooper starting sufficiently far from the aircraft so that the subcomputational walls both avoid the aircraft and reside fully in fluid elements to project the data. This applicational concern will arise and must be noted.

5.2.2 Outflow Boundary Conditions

Then there arises the problem of the size of the subcomputation. If the subcomputation is modeled too largely, then the length of the path or trajectory is limited because this larger subcomputation must travel a path in the main flow field. In the case of my paratrooper exiting an AirForce cargo aircraft, the domain must follow the paratrooper from the aircraft door (as close to it as possible) to the domain walls. A larger subcomputational domain will hit the main domain’s walls sooner because it is larger, so a shorter trajectory will be computed. If the subcomputation domain
is too small or too close to the separating object, then the subcomputational outflow boundary will not be stress–free, and the boundary stress component on the right hand side of Equation 3.12 will not be zero and must be calculated. In the calculations presented here, this normal component of stress is calculated on part of the boundaries of the subcomputational domain since the walls are close to the object to allow a greater trajectory.

5.2.3 Conflicting Boundary Conditions

Usually computational domain walls are rectangularly aligned with the Cartesian coordinate system. But the separating object is allowed to rotate freely. As the object's tracked orientation rotates, so does the boundary conditions on the fixed subcomputation. Consequently, many times there will be nodes on a given subcomputational boundary that have flow coming into the domain and nodes with flow leaving the domain. Some care must be given to decide how to determine boundary types for FOIST. The scheme used here utilizes threshold values.

1. If a given boundary has inflow nodes that constitute a certain threshold percentage of the total nodes on that boundary or higher, the boundary is labeled inflow. And the nodal values are set, even the value of the nodes with outflow, as Dirichlet boundary conditions.

2. If the inflow nodes are below that threshold value, then the boundary is tested to determine if the outflow nodes constitute a certain threshold percentage of the total nodes on that boundary or higher. If so, the boundary is labeled outflow and the normal component of stress will be calculated for all nodes, both inflow and outflow nodes, on that boundary. If the outflow wall is sufficiently far away, as aforementioned, the boundary will be stress–free.

3. If neither of the above conditions are true, then if a threshold percentage or higher of the boundary nodes have neither an inflow or outflow velocity but have a zero or planar velocity, the boundary is labeled slip. And all nodes
whether they had an inflow, outflow, zero, or slip velocity, are assigned slip conditions.

4. If none of the above conditions are true, then if a plurality of nodes are inflow, the boundary is labeled inflow. I always choose to err on the side of making nodes with outflow velocities have an inflow condition rather than vice-versa. This is due to the fact that inflow boundary conditions will still set the nodes with outflow velocity to their actual values. But the inflow information is lost when the node with inflow velocities are on a boundary labeled outflow.

5. Otherwise, the boundary is labeled outflow.

The thresholds are different for different simulations. They are usually set to create similar boundary conditions as the main flow field before the subcomputational domain should experience any rotation.

5.3 Applications

I will now look at two simple test cases. The first involves a ball in uniform flow. The second simulates a ball in shear flow. And then the trajectories will be compared to mesh moving control cases to see how strong and effective an approximation is FOIST. For the control problems, the DSD/SST procedure was used for the fluid, and the equations of linear elasticity were used to move the mesh. Unstructured triangular meshes were used for the fluid domain with occasional remeshing to avoid severe mesh distortion. The simulations were carried out on the CrayT3E parallel supercomputer.

In both simulations the ball has a radius of 1 model length unit, a mass of 5.0 model mass units. The viscosity is 3.24e-8 squared model length units per model second units.

Here, again, I have included the pictures of the ball from Chapter 4 which portrayed special mesh design. Except for the method, the cases were to be as close to
the same as possible. So, in order to minimize the difference between the meshes a fixed inner refinement zone was created for the ball for the mesh moving case. This zone would move with the ball and distortion would only occur outside this fixed region. This same refinement zone would exist around the ball for the subcomputation in FOIST. This fixed refinement zone served to give equal refinement to both the FOIST case and the control. You can see the inner refinement zones in the first two pictures of Figures 5.1 and 5.2. The third picture in the figure shows the inner moving zone for the mesh moving control problem to be compared to FOIST.

5.3.1 Ball in Uniform Flow

For the first case, a sphere was placed in a uniform flow, and it was only allowed to translate. Any calculated rotational displacement was canceled, or set to zero. The inflow velocity is set to 1 model length per model time.

Below in Figure 5.3, the comparison between the mesh moving case and the FOIST case is presented. It can be seen that the mesh moving control case presents the ball with a greater drag force. In the FOIST case, the ball falls more quickly and is not swept back as much.

The difference in trajectory is due to the close walls of the subcomputational domain around the ball in the FOIST simulation. Since the flow may not truly be slip on the side walls or even uniformly 1 on the inflow (at that distance) it creates a change in the force the ball would feel if it were actually in the main flow field. And this force change can be noted in the graph. This effect is lessened when adding rotation.

5.3.2 Ball in Shear Flow

In this example, the same ball is subjected to shear flow. This flow is linearly distributed on the inflow boundary of the main flow from 0 to 1 model length per model time units as one traverses the inflow boundary in the positive y-direction, opposite to the direction of gravity. Rotation is included in these simulations.
Figure 5.1. 3D Surface Meshes of Flow Around a Ball.

Now, in this example as in the usual FOIST examples, since the object is allowed to rotate, any subcomputational boundary can become an inflow, outflow, or slip boundary, as the boundary conditions are updated each time step. So the domain walls of the subcomputation are equally spaced from the object since any wall can be
outflow or inflow.

In Figure 5.4, it can be noted that FOIST gives a better approximation under the true conditions of both translation and rotation. The error of the FOIST approximation from the mesh moving case is still partly due to the closeness of the subcomputational boundary walls to the object in question, here a ball. But now there is a part due to the approximation of the boundary condition as the ball rotates and its invisible subcomputational domain rotates in the main flow field. With each time step and each translational and rotational displacement, the orientation and location of the nodes on the boundary of the subcomputation in the main flow field change. And with many orientations, some of the subcomputational boundaries will
have a mixture of nodes with inflow velocities and nodes with outflow velocities. In labeling the entire boundary as outflow, inflow, or slip, some information is lost and a resulting change in applied forces is felt by the ball. This can cause both too much force and too little force, or force felt in the wrong direction. So this new error combined with the previous error accounts for the discrepancy. But FOIST approximates much better with both translation and rotation occurring. To see other directional comparisons, refer to Figures A.1 and A.2 in A.

More test cases can be found in Chapters 7 and 8. There I can see the effectiveness of FOIST in even more complex problems. I especially want to observe its approximating power when there is a larger object in the main flow field that effects the main flow field and provides gradients in the velocity flow field. There FOIST can be tested for its ability to model separating problems, a focus for this work.
Figure 5.4. Trajectory of Ball in Shear Flow: FreeStream vs Gravity Direction.
Chapter 6

Boundary Layer Resolution

Whether using the DSD/SST method with mesh moving or using a mesh moving alternative like FOIST, there is still the problem of the thick numerical boundary layer around objects. This thick numerical boundary layer is not physical, but rather a result of poor resolution on the surface of the objects over which the flow is being calculated, such as an AirForce cargo aircraft. When calculating the flow around such an object, alone, the problem is not too burdensome. But in combination with a separating smaller object like a cargo payload or a jumping paratrooper, a non-physical, excessively thick numerical boundary layer can artificially effect the path of such a separating object. And so the relevance and importance of resolving the boundary layer is ostensibly seen.

In the sections of this chapter, I will attempt to minimize and eliminate the boundary layer of an AirForce cargo aircraft through two methods. In Section 6.1 the addition of the new capability of boundary layer elements to my 3D mesh generator will be presented and its effect will be analyzed. And in Section 6.2 physically thin boundary layers will be approximated through slip conditions newly implemented and later applied to my 3D paratrooper–aircraft simulation.
6.1 Boundary Layer Elements: Overview of Current Problems

6.1.1 Overview of Current Problems

Boundary layer elements are an important topic in the modeling area of computational fluid dynamics. In solid dynamics, whenever there are areas of relatively high displacement, increased refinement is required to resolve those areas properly. The same is true in fluid dynamics. When there are areas of high circulation or vorticity, for example, more elements are needed to capture such a flow with high velocity gradients.

The problem arises with current 3D mesh generators. The algorithm used by most 3D mesh generators distributes increased refinement throughout a 3D space in an uneven way. To clarify, imagine a domain in the shape of a rectangular prism. Inside the domain is an AirForce cargo aircraft. Let's assume a 3D surface mesh has been created on both the aircraft and the domain walls, so each is filled with triangular elements and nodes. And I want to create a 3D fluid mesh between the aircraft and the domain walls. If the refinement on the walls is equal to the refinement on the aircraft, then the tetrahedra in the 3D space will roughly be the same everywhere, from the aircraft to the domain walls. Now, if increased tetrahedra are desired on the surface of the aircraft, the refinement on the surface of the aircraft can be increased, as seen in Figure 6.1 contrasting with the larger refinement on the surface of the domain box shown in Figures 6.2, 6.3, and 6.4.

The mesh generator will take the increased refinement on the aircraft to create smaller and more tetrahedra at the surface of the aircraft. The logical deduction is that the size of the tetrahedra will change roughly linearly or at least evenly between the small tetrahedra on the surface of the aircraft and the large tetrahedra that will border the domain walls. But most automatic 3D mesh generators will favor the large tetrahedra corresponding to the larger refinement on the domain walls. In other
words, more of the tetrahedra in the 3D fluid mesh between the aircraft and the
domain walls will be closer to the size governed by the refinement on the domain
walls than that governed by the refinement on the aircraft.

Another problem with a simple refinement increase on the object with a boundary
layer is that many times the refinement elsewhere in the mesh is perfectly acceptable.
Only the number of elements at the surface of the object need be increased. So, when
increasing refinement on the object in question, the increased elements throughout
the fluid mesh can be wasteful as only the extra elements in close proximity to the
surface of the object are needed or desired. From the use of the automatic 3D mesh
generator with the given refinement shown in Figure 6.1, the resulting numerical
boundary layer can be seen in Figure 6.5. And yet, after doubling the refinement and
incurring extra elements away from the surface of the aircraft, little change can be
seen from this general increased effort, Figure 6.6.

So boundary elements are both useful and cost efficient in that they provide in-
creased elements only at the surface or boundary needing refinement. My automatic
3D mesh generator, developed by Dr. Johnson [18], does not have the capability to
create boundary layer elements. It is possible to manually create boundary layers on
boundaries aligned with the Cartesian coordinate system, but the creation of bound-
ary layers on arbitrary geometries is tough area of research that Dr. Johnson is still
currently researching. In Appendix A, Figures A.3-A.8. explicate how thickness of
the numerical boundary layer affects the speed of the air hitting the paratrooper.
6.1.2 Boundary Layers: New Contributions

In this thesis, two new companion programs were created to act on meshes after they have been generated by the automatic 3D mesh generator. These programs add 3D boundary layer tetrahedra to arbitrarily shaped boundaries, and they simply represent two variants of boundary layer creation. This contribution is significant for two reasons. First, no programs produce tetrahedral boundary layer elements. Secondly, none produce boundary layer elements on arbitrarily shaped surface geometries. Usually if such a tetrahedral boundary layer mesh is needed on an arbitrarily
shaped surface, it must be generated manually. These two programs now do this automatically.

A layer of tetrahedral elements must first be defined. A layer of tetrahedral elements on a surface or boundary is all tetrahedra having a face or a point on the specified boundary or face. A layer above that initial layer would include all tetrahedra bordering, either by a point or face, all tetrahedra in the first layer. The effects of both variant programs on the boundary layer will be observed.

**Even Boundary Layers**

The first variant of my new boundary layer program creates evenly sized layers of tetrahedra on the surface of the specified boundary. Of course, the costs increase with increased number of elements. And so it is infeasible to continually increase the layers of boundary elements without end, as the aircraft is a very large object with a
Figure 6.4. 3D Surface Mesh of Highlighted Aircraft and Domain Box: Side View.

very small refinement on its surface. And so three different trials were attempted.
Figure 6.5. Color Plot of Velocity in Boundary Layer.
Figure 6.6. Color Velocity Plot in Boundary Layer with Double Refinement on the Aircraft Surface.
Figure 6.7. Boundary Layer Around Fuselage of Cargo Aircraft.
Figure 6.8. Boundary Layer Velocity Surface Around Fuselage of Cargo Aircraft: Side View.
Those three trials were a two-layer tetrahedral boundary layer, a four-layer tetrahedral boundary layer, and an eight-layer tetrahedral boundary layer. The resulting boundary layers are viewable in Figures 6.7 and 6.8. Upon first inspection, it seems that the boundary layer elements make little or no difference. There is a small decrease in the boundary layer, but the decrease seems imperceptible. Decreasing the numerical boundary layer around the cargo aircraft becomes a difficult task because as you move from the nose of the aircraft to the tail, the perimeter and effective diameter of the fuselage increase as I encounter extensions or protrusions to the fuselage. This creates areas of decreased velocity as the air flow must come to a stop or point of zero velocity at additional stagnation points. So the small decrease in the numerical boundary layer is resulting from the enlarging shape of the fuselage. Otherwise, boundary layer elements have been proven to effectively increase the resolution of the boundary elements when the physical flow actually experiences only a very thin layer, [35], [18].

Additionally, the cargo aircraft is flying at an angle of attack of 8°, so separation occurs sooner on the top side of the cargo aircraft than if it were flying at a 0° angle of attack. This creates a slightly larger boundary layer on the top side of the fuselage than on the bottom side.

**Boundary Layers with Internal Growth**

The second new program creates layers of tetrahedra with a growth factor of 2 moving away from the plane into the fluid mesh towards the domain walls. So, if the first layer of boundary elements are attached to the surface of the cargo aircraft, then elements in the second layer of boundary elements have roughly twice the element length of those in the first layer of boundary elements.
Figure 6.9. Boundary Layer Velocity Surface Around Fuselage of Cargo Aircraft.
Figure 6.10. Boundary Layer Velocity Surface Around Fuselage of Cargo Aircraft: Side View.
As seen in Figures 6.9 and 6.10, the same problems occur. A considerable decrease in the size of the numerical boundary layer is not observed. And the same problems seen in in the previous subsection about even boundary layers are evident here, as well. The same explanations apply, too. So I expect boundary layers to work in objects with constant diameter such as a 3D cylinder. But the same difficulties arrive with a growth approach in the boundary layer for the fuselage of the cargo aircraft.

In cases when the boundary layer is expected to be very thin, thinner than the numerical results show, and the Reynolds number is very high, I can approximate the flow with slip conditions on the specified object. This provides another option or means of reducing and possibly eliminating the numerical boundary layer.

6.2 Slip Boundary Condition

Employing slip boundary conditions allows the velocity to slip, stress–free, across the arbitrarily shaped boundary. The numerical boundary layer should thus be severely reduced or eliminated. And my belief is that the effects should be more pronounced then the use of boundary layer elements. First, the implementation will be discussed referencing Le Beau’s work [36], and then its effectiveness on the numerical boundary layer of the cargo aircraft will be examined.

6.2.1 New Contribution: Implementation of Slip Boundary Condition

A slip boundary condition arises when the normal component of the velocity of a boundary is set to zero and the tangential components are assigned stress–free conditions. This allows the fluid particles to slide along the surface of the boundary in a stress–free state. This is exactly the condition I wish to utilize for eliminating the boundary layer of the cargo aircraft.

In the past, this was first done explicitly. The killing of the normal component of velocity was attempted. The rotation of velocity vectors while maintaining the
kinetic energy of the fluid particles was also implemented as an explicit means. The
drawback is the decrease in the stability of the algorithm due to this explicit killing
of the normal component.

In order to maintain the stability of the algorithm, an implicit method is chosen
here. Following Le Beau’s structural outline for 2D slip boundary conditions using the
Euler equations [36], I recall that, in the finite element formulation, velocity within
an element is the sum of the value of the shape functions for each node multiplied by
the velocity value for that corresponding node. Thus, the velocity can be represented
by

\[ U = \sum_{B=1}^{n} N^B U^B, \]  \hspace{1cm} (6.1)

where \( U^B \) and \( N^B \) represent the velocity and shape node function values at node \( B \),
respectively, and \( n \) represents the number of elemental nodes. For a three-dimensional
tetrahedral element, this can be expanded in vector form to show

\[
\begin{pmatrix}
U^1 \\
U^2 \\
U^3 \\
U^4
\end{pmatrix}
= \begin{pmatrix}
N^1 \\
N^2 \\
N^3 \\
N^4
\end{pmatrix}
\begin{pmatrix}
U^1 \\
U^2 \\
U^3 \\
U^4
\end{pmatrix}
\]  \hspace{1cm} (6.2)

where the superscripts represent the degrees of freedom of the unknown. Those un-
knowns are the three components of velocity, \( u, v, \) and \( w \), in the Cartesian directions
and pressure \( p \).

In some cases, on the boundary, when velocities or stresses are specified (Dirichlet
or Neumann boundary conditions), the shape of the boundary is arbitrary and it is
not aligned with the Cartesian coordinate frame. In such cases, a nodal vector is
defined within the reference frame formed by normal and tangential vectors at that
point such as
\[ \mathbf{d} = \begin{pmatrix} u_x \\ u_\psi \\ u_n \\ p \end{pmatrix}, \quad (6.3) \]

where \( u \) represents velocity, \( n \) signifies the normal component at that node, and \( \chi \) and \( \psi \) are the orthogonally tangential components at the given node. Now, that the coordinate system is in the local frame of the node, velocity values can be set directly for Dirichlet boundary conditions. In practice, the normal component, \( u_n \), is typically assigned a value for such conditions.

Now, the problem lies in having multiple nodes with velocity vectors in their local coordinate systems. Probably most nodes in any given simulation will have vectors in the Cartesian coordinate frame. But any nodes on slip boundaries with arbitrary shapes will all have vectors of unknowns in their local coordinate systems. In order to solve the system of equations for the unknown velocity vectors, it is necessary to have all nodal vectors in the same coordinate frame. Thus there is a need to transform the nodal vectors in various local coordinate systems to the Cartesian coordinate frame. Then the governing equations can be solved for the unknowns. Equation 6.2 can be written to show

\[ \mathbf{U} = \begin{pmatrix} U_1^1 \\ U_1^2 \\ U_1^3 \\ U_1^4 \end{pmatrix} N^1 + \begin{pmatrix} U_2^1 \\ U_2^2 \\ U_2^3 \\ U_2^4 \end{pmatrix} N^2 + \mathbf{T}^3 \begin{pmatrix} d_3^1 \\ d_3^2 \\ d_3^3 \\ d_3^4 \end{pmatrix} N^3 + \mathbf{T}^4 \begin{pmatrix} d_4^1 \\ d_4^2 \\ d_4^3 \\ d_4^4 \end{pmatrix} N^4, \quad (6.4) \]

where nodes 3 and 4 lie on the boundary and \( \mathbf{T}^i \) represents a rotation matrix that rotates the local coordinate frame of node \( i \) to the Cartesian coordinate reference. In 3D, the rotation matrix \( \mathbf{T} \) should be a matrix including rotations about multiple axes to align the local coordinate axes with the Cartesian axes. So there will usually be multiple angles of rotation. Now, rewriting Equation 6.1 for boundary nodes I find
\[ U = \sum_{B=1}^{n} N^B T^B d^B. \]  

This rotation must also effect the trial functions in the same manner, generally defining the trial function, \( W \), as

\[ W = \sum_{A=1}^{n} N^A c^A. \]  

For slip boundary nodes on arbitrary surfaces, the equation is modified to include the rotation matrix; thus,

\[ W = \sum_{A=1}^{n} N^A T^A c^A. \]  

These new definitions in Equations 6.5 and 6.7 can be generalized so that for nodes that are not on slip surfaces, the matrix \( T \) represents the identity matrix and the vectors \( c \) and \( d \) are already in the Cartesian coordinate reference frame. With that generalization I can substitute those definitions in my DSD/SST finite element formulation found in Equation 3.12, ignoring the stabilization terms for now, to obtain

\[
\int_{Q_n} \left( \sum_{A=1}^{n} N^A T^A c^A \cdot \rho \sum_{B=1}^{n} \frac{\partial N^B}{\partial t} T^B d^B \right)
+ \sum_{A=1}^{n} N^A T^A c^A \cdot \rho \left( \sum_{B=1}^{n} \frac{\partial N^B}{\partial k} T^B d^B \right) \sum_{B=1}^{n} N^B T^B d^B
+ \sum_{A=1}^{n} N^A T^A c^A \cdot f^h
- \sum_{A=1}^{n} \frac{\partial N^A}{\partial x_i} T^A c^A \sum_{B=1}^{n} N^B p^B
\]
\[ + \mu \left( \sum_{A=1}^{n} \sum_{B=1}^{n} \left( \delta_{ij} \frac{\partial N^A}{\partial x_k} T^A c^A \cdot \frac{\partial N^B}{\partial x_k} T^B d^B_j + \frac{\partial N^A}{\partial x_j} T^A c^A \cdot \frac{\partial N^B}{\partial x_i} T^B d^B_k \right) \right) dQ \]

\[ + q^h \sum_{B=1}^{n} \frac{\partial N^B}{\partial x_j} T^B d^B_j dQ \]

\[ + \int_{\Omega_n} \left( \sum_{A=1}^{n} N^A T^A c^A \cdot \rho \sum_{B=1}^{n} N^B T^B d^B \right) d\Omega \]

\[ - \sum_{A=1}^{n} N^A T^A c^A \cdot (u^h)_n^- d\Omega \]

\[ = \int_{(P_n)_h} \sum_{A=1}^{n} N^A T^A c^A \cdot h^h dP. \quad (6.8) \]

where the superscript \( i, j, \) and \( k \) refer to degrees of freedom within each nodal block, while \( d^B \) and \( p^B \) represent the velocity and pressure components of the unknown vector, respectively. Since the \( c^A \)'s are arbitrary, this can be rewritten as

\[ \sum_{B=1}^{n} \int_{Q_n} \left( N^A (T^A)^T T^B \rho \frac{\partial N^B}{\partial t} \right) d^B \]

\[ + N^A (T^A)^T T^B \rho \left( \sum_{B=1}^{n} \frac{\partial N^B}{\partial k} T^B d^B_k \right) N^B d^B \]

\[ + N^A (T^A)^T f^h \]

\[ - \frac{\partial N^A}{\partial x_i} (T^A)^T N^B p^B \]

\[ + \mu \left( \delta_{ij} \frac{\partial N^A}{\partial x_k} (T^A)^T T^B \frac{\partial N^B}{\partial x_k} d^B_j + \frac{\partial N^A}{\partial x_j} (T^A)^T T^B \frac{\partial N^B}{\partial x_i} d^B_j \right) dQ \]
\[ + q^h \frac{\partial N^B}{\partial x_j} T^B d_j dQ \]

\[ + \int_{\Omega_n} \left( N^A(T^A)^T T^B \rho N^B d_B^+ \right) \]

\[-N^A(T^A)^T (u^h)_n^- \int_{(P_n)_h} \]

\[ = \int_{(P_n)_h} N^A(T^A)^T h^h dP. \quad (6.9)\]

The nonlinear advective term will not be solved in separate terms where one unknown is frozen and treated as known while the other is treated as a variable unknown. So in practice, two summations would not be completed in that term.

Interestingly, such rotations could considerably increase the computational costs, so means to reduce the rotational costs become important in such slip boundary problems. Since the majority of nodes in a given simulation are not slip boundary nodes, the rotations do not need to be employed for every element. Utilizing the rotations only when needed is the first cost-saving method. Secondly, the rotations need not occur until after the entire formation. The rotations can be moved outside the summations, outside the entire element level loop, to avoid rotating every term inside the elemental and inner shape function loops. This saves time and also allows for the vectorization of the element level loops.

It should also be noted that another new contribution in the implementation of the slip boundary condition using Navier–Stokes equation for incompressible flow is in the calculation of the 3D point–normals. On a particular point that is the intersection of multiple triangles (sides of tetrahedra) in space, the normal is usually calculated by weighting the normals of the various touching triangles. The weighting factor of a normal of a particular triangle is usually

\[ w_f = \frac{A}{A_t}, \quad (6.10)\]

where \( A \) is the area of the triangular side and \( A_t \) is the total cumulative area of all triangular sides at that point.
In my implementation, I also use a weighting factor based on the angle at that point. So regardless of the area if the angle is large at the point it largely affects the direction of the point–normal. This presents a new weighting factor equation,

\[ w_f = \frac{\alpha}{\alpha_t} \quad (6.11) \]

where \( \alpha \) is the angle made by the corner of the triangle in question at the specific point and \( \alpha_t \) is the total sum of all angles made at that point by bordering triangles. The normals at a particular point are then calculated by summing the weighted normals of the bordering triangles.

### 6.2.2 Slip Boundary Applications

Finally, the effects of the 3D slip condition on an arbitrary boundary must be validated. Before observing its influence on the boundary layer of the cargo aircraft, I choose to observe its effect on the flow around a ball.

Using a slip boundary condition on the surface of the ball, Figure 6.11 clearly displays the boundary layer observed around the ball.

It is nonexistent or, at least, imperceptible. This is what is expected. But I must also attest to the accuracy. Compared with an analytical solution for inviscid flow about a sphere, the norm of the error vector (vector containing the error for every node in the simulation) is 0.001. So obviously, the slip code can approximate an inviscid simulation about a sphere very well. And I am satisfied with the error. The error that is observed, is simply due to imperfection of the modeling of the sphere and the lack of symmetry. If the refinement is increased, the error decreases. Repeating the same simulation with twice the refinement gives an error norm of 0.00001. This is an improvement that demonstrates the source of the error is as conjectured.

Now that the slip boundary condition code is validated, its effectiveness can be tested on the cargo aircraft problem in order to see if it helps solve the thick boundary layer issue. In Figure 6.12, I can see that there is no noticeable boundary layer. And
such a slip implementation can be used with objects separating from the aircraft to avoid the negative interaction of a false, numerically thick boundary layer.
Figure 6.12. Velocity in Free–stream Direction About Cargo Aircraft with Slip Boundary Conditions.
Chapter 7

Paratrooper Separating from an AirForce Cargo Aircraft

In this chapter, I present the FOI simulations of a paratrooper jumping from the cabin of an AirForce cargo aircraft. Initially, these simulations were started by Johnson using a plethora of mesh moving methods including DSD/SST [18]. And those simulations are now continued here. All simulations presented are new works con-

Figure 7.1. Paratroopers Preparing to Exit from Cargo Aircraft.
Figure 7.2. Cargo Aircraft in Formation with Paratroopers Exiting.

tributed by myself. When the project was passed to me, the paratrooper was falling as shown in Figure 7.9 and 7.10. Unless otherwise marked as old (i.e. old models), all other figures graphs, methods, and results are new contributions developed and simulated as new work in this thesis focusing on the problem of paratrooper–aircraft separation.

The interest in the problem is manifold. With this cargo aircraft and similar others, the US Army has encountered the problem of crossover: the phenomenon whereby from opposite doors of the same aircraft, two simultaneously jumping paratroopers cross paths as they fall below the plane. Of course, this is is potentially injurious, and a solution is crucial. In such cases, CFD proves its worth as a faster, cheaper, and, especially here, more convenient alternative than an experimental test. I would not want lives put at stake to experimentally explore the causes and ramifications of crossover.

Secondly, when these planes fly in formation, all succeeding aircraft fly at a higher altitude than the preceding aircraft, Figure 7.2. Fatalities, or casualties, have occurred when paratroopers jumped and fell through the wake of preceding aircraft. The vortices and resulting turbulence from the preceding aircraft can be quite violent and prevent a paratrooper from properly pulling the cords to deploy his parachute. So the price of such a study is even more desirable.
Besides the basic aerodynamic forces, there are forces that can influence the trajectory of the paratrooper. First, meteorological conditions can alter the flow of air and path of the paratrooper. That is why it is important for a simulation to be done with different types of conditions, as is the case in real physical life. Secondly, the geometry of the plane has some influence over the path of the paratrooper. This occurs because the geometry of the plane can direct the path of the air as it flows about the aircraft. That in turn can effect the paratrooper trajectory.

I now continue the work focusing on the crossover problem and the first 2–3 seconds after exiting the cargo aircraft. The aim is to observe a force that tends to push the paratroopers across the symmetry plane of the aircraft and to simulate crossover. However, my ultimate goal is to make paratrooper jumps from such aircraft safe and effective. If a real, accurate trajectory of the paratrooper can be simulated, then geometry changes can be suggested that will beneficially effect the airflow and thereby the paratrooper paths. Once that is completed, then meteorological conditions such as cross wind can be introduced to analyze their effect and test if accurate trajectories can still be modeled in such weather patterns.

7.1 Fluid Model

In modeling the cargo aircraft, I assume symmetry with respect to the plane passing through the middle of the aircraft. And so I model only half of the aircraft for this simulation of a jumping paratrooper exiting the cargo aircraft. The original aircraft model was introduced by Johnson, [37] seen in Figure 7.3.

Now, in this work, wing flaps and wing tips were added constructing a more accurate model. This can be seen in Figures 7.4 and 7.5.

The model update was performed in order to improve the results and accurately simulate the trajectory of the jumping paratroopers. By more accurately modeling the aircraft, a more accurate air flow can be computed about the aircraft. And this air flow helps determine the path of the paratrooper.

But the paratrooper model, itself, needed to be updated, as well. Previously
called Oscar due to its Academy-Award resemblance, the older model by Johnson, had a disproportionate volume relative to its height, shown by Figure 7.6. A real paratrooper, as was seen in Figure 7.1, is weighted by his equipment and gear that adds to his volume. Even without that, the average paratrooper’s volume is increased by his bulky uniform. Without a proportionally correct volume, the paratrooper has always suffered from a relatively high density. So an increase in the accuracy of the simulation should result from an update of the model of the paratrooper.

Front and side views of the paratrooper can be seen in Figure 7.7. It should be noted that this paratrooper has no parachute pack on his back. This particular model is used when simulating later stages in the jump of the paratrooper such as the time when the paratrooper falls through the wake of a preceding aircraft. At that point
Figure 7.4. Updated Cargo Aircraft Surface Mesh.

Figure 7.5. Updated Cargo Aircraft Surface Mesh—Posterior View.

the parachute pack has fully opened. This work here focuses on the first 2–3 seconds of the paratrooper jump, and so the model used for this work is displayed in Figure 7.8.

Three-dimensional triangular surface meshes were then created from the models,
and then a 3-D tetrahedral volume mesh was generated for the fluid dynamics solution using these surface mesh. The mesh consists of 129,090 nodes and 728,902 tetrahedral elements. The modeling software, 3D surface mesh generator, and automatic 3D mesh generator were all developed by Dr. Andrew Johnson [24].

7.2 FOI Setup

In this example of an AirForce cargo aircraft, the aircraft is traveling at 130 knots,
or 66.88 m/s. This is a typical cruising speed for such drop missions like paratrooper jumps or cargo payload drops. The aircraft flies with an angle of attack of 8 degrees. I assign the following boundary conditions for the FD model:

- The cargo aircraft and paratrooper surfaces are assigned a prescribed boundary condition for all components of velocity. On these surfaces, I achieve a no-slip condition by prescribing zero velocity.

- The inflow boundary in front of the aircraft is assigned a prescribed boundary condition, with the velocity components representative of the free-stream air-speed.

- The crossflow and side boundaries are assigned free-slip conditions, with no normal flow permitted through the boundaries.

- The outflow boundary behind the aircraft is assigned a traction-free boundary condition.

In defining the parameters of the problem with non-dimensional numbers, the length of the aircraft (nose to tail wingtip) is taken as 8.81 units, and the free-stream velocity as 1 unit. For the engines, the intake and exhaust flow velocity profiles are assumed to be uniform, and at 1.0 and 3.0, respectively.

### 7.3 FOI Simulation

In attempting to correct the unreal, previously simulated trajectory with the Oscar model, Figure 7.9, not only were the models updated, but among other corrections the falling paratrooper was now treated as a 3D object and given a 3D mass moment of inertia. Previously treated as a sphere with an inertia vector with three directional components, the error contribution may have been considerable (Figure A.9 in Appendix A). Figure 7.10 displays the trajectory in one layered picture.

With the many updates and corrections, a true, viable trajectory was finally simulated. First, the pressure distribution on the aircraft during the simulation can
be viewed in Figure 7.11. The jet engines are colored blue, and that blue does not represent pressure. Figure 7.12 shows the turbulent flow the paratrooper must jump into. The figure gives a picture of the streamribbons that flow about the aircraft during a snapshot in the simulation. And the presence of the aircraft also affects the flow about the paratrooper. The pressure distribution on the surface of the paratrooper can be seen in Figure 7.13. The figure also displays the streamlines flowing about him. Produced after running the corrected simulation with the new and improved models, the new trajectory is shown in Figure 7.14.

Now, my attention is turned to the validation of this simulated trajectory. How does it compare to that of real trajectories? There was no explicit information available to me during this research, so large estimations were made to approximate the distance and slope of real trajectories from the videotape of paratroopers jumping from an AirForce cargo aircraft. A comparison is now made between two of those real trajectories, my simulated trajectory, and an analytical trajectory.

From the many hundreds of videotaped jumps, two real trajectories were chosen as extremes. One was the trajectory with one of the paratroopers seemingly most swept back, or experiencing the most drag. And the other followed one of the paratroopers who was seemingly least swept back, experiencing the least amount of drag. The analytical trajectory was calculated using an estimated coefficient of drag–2.3. Using equation 7.1, the drag force is calculated on the paratrooper,

\[ F_D = C_D \rho V^2 A, \]  

(7.1)

where \( F_D \) and \( C_D \) are the drag force and coefficient of drag, respectively, \( \rho \) is the density, \( V \) is the speed of the flow, and \( A \) is the projected surface area exposed to the flow. The speed of the airflow is taken from the numerical simulation which is 66.8 m/s. Summing with the gravitational force, a net force is calculated from which an acceleration is derived. This is then integrated once and twice to produce the velocity and the new displacement.

Figure 7.15 compares the real trajectories to the analytical and numerical ones. The slopes of the analytical, numerical, and two real trajectories are -0.586, -1.687,
-1.141, and -0.713, respectively. These slopes were measured only using the starting and ending points of each trajectory. The analytical trajectory was swept back the most. It can be observed that the trajectory produced from the numerical simulation is a possible, viable trajectory when compared to the real trajectory of the one of the paratroopers least swept back. Because the paratroopers’ jumps are so different and erratic, much freedom is allowed in the description of an “average” jump. In other words, there is no one body configuration noted in all jumps and definitely no one jump trajectory. Therefore, since the trajectory from the numerical simulation is close to the that of one of the real trajectories, it is a possible trajectory.

Still, through a numerical simulation, it should be possible to produce the real trajectory of one of the paratroopers who is most swept back. Believing this is due to a numerically thick boundary layer, especially since the analytical trajectory is more swept back than the numerical one. I now apply the slip method in Chapter 6 to hopefully improve the solution.

### 7.3.1 Slip Simulation

Here I attempt to simulate the same separation problem of the jumping paratrooper exiting from the cargo aircraft with slip conditions. The boundary conditions are the same as stated in Section 7.2 with the one change applying to the surface of the aircraft. It is now given a slip condition allowing the fluid particles to flow along the surface of the aircraft, thereby eliminating the boundary layer and the low drag problem. Figure 7.16 shows this new trajectory.

And now I add this new trajectory in my comparison to the real videotaped trajectories. Figure 7.17 clearly compares all trajectories and validates the numerical results. Using the slip approximation, clearly, this code has risen to prove new possible, viable, and true trajectories for the high drag experiences of real paratroopers. This is an important contribution.
7.3.2 FOIST Simulation

Lastly, in this chapter, I wish to newly examine the effects of the new FOIST method on such a complex 3D problem as the aircraft-paratrooper separation problem. And my desire is to determine if the trajectory is approximated well enough that the cost savings are justified.

First, a subcomputational domain was modeled and constructed containing only the paratrooper. This is displayed in Figure 7.18. Then a main flow was simulated around the aircraft without the paratrooper assuming, again, that the influence of the paratrooper on the main flow field is negligible.

Then, using the same specifications and conditions outlined in Section 7.2, the FOIST subcomputation around the paratrooper was simulated using projected data from the main flow field, Figure 7.19.

Ostensibly, the new FOIST approximation does not match perfectly with the mesh moving case. This is due to the experience of less drag, the reasons of which were explained in Chapter 5. But, here, the approximation is not too poor and is satisfactory in presenting a general idea of the shape and extent of the paratrooper trajectory. This can be a helpful tool in analytical and directive preprocessing. And this is exactly the type of tool to be utilized in the predictive prognostication of the trajectory tendencies of separating objects. Also, as the simplicity of the problem increases, so does the approximating power of FOIST.
Figure 7.7. Updated Paratrooper Model.
Figure 7.8. Updated Paratrooper Model with Parachute Pack.
Figure 7.9. Trajectory Snapshots of Original Paratrooper Model.
Figure 7.10. Old Trajectory of New Paratrooper Model.
Figure 7.11. Pressure Distribution on Cargo Aircraft.
Figure 7.12. Streamribbons about Cargo Aircraft Colored with Pressure.

Figure 7.13. Pressure Distribution on Paratrooper with Streamlines.
Figure 7.14. New Trajectory Snapshots of New Paratrooper Model.
Figure 7.15. Paratrooper Trajectory Comparison: Real vs Numerical vs Analytical.
Figure 7.16. Paratrooper Trajectory with Slip Conditions on Aircraft.
Figure 7.17. Paratrooper Trajectory Comparison: Real vs Numerical vs Analytical vs Slip.

Figure 7.18. Model of FOIST Subcomputational Domain Containing Paratrooper.
Figure 7.19. Paratrooper Trajectory Comparison: Real vs Numerical vs Analytical vs Slip vs FOIST.
Chapter 8

Cargo Payload Separating from an AirForce Cargo Aircraft

Now my attention is turned to Fluid–Object Interaction simulations of a payload falling from the cargo bay of an aircraft. This is another separation problem. And the same new techniques and methods employed for the jumping paratrooper simulation can be apprehended for this problem of cargo drops. Moreover this is an entirely new application, and all figures, charts, simulations, tables, and this entire chapter constitute new work and research and provide new contributions.

The CFD solution to this similar problem satisfies many interest for the US Army. In general, the trajectory of such drops is important for the prediction of the landing point of the exiting objects. And these objects can take many forms.

For instance, if a CIA operative in Afghanistan is in need of a bundle of cash for bribery, such cargo drops could be utilized. If the landing point of the bundle of cash can be pinpointed within a 500 yard diameter, the operative can locate the bundle. So the importance of such simulations is readily apparent.

Again, I continue the work focusing on the first 2–3 seconds of the cargo drop after exiting the cargo aircraft. The aim is to observe and simulate real trajectories in an effort to predictively determine the future paths of similar objects to be deployed from such cargo aircraft. And this information can then be adapted for various military
missions.

8.1 Fluid Model

In modeling the cargo aircraft for use with payload drops, the location of the cargo door must be taken into account. It is located along the symmetry line of the plane in the center of the bottom of the aircraft fuselage. For this reason, in order to simulate
payload drops directly from the center of the cargo door, the entire cargo aircraft must be modeled. Two views of the newly designed and created cargo aircraft are shows in Figures 8.3 and 8.4.

Figure 8.3. New Cargo Aircraft Model with Cargo Bay.

Figure 8.4. Cargo Aircraft with Cargo Bay–Side View.

Next, a cargo prototype was found, Figure 8.5. Since no specific prototype was needed as long as it fit within the cargo aircraft, this first picture found was used. It is interesting to note the hole in the middle of the payload body. Perhaps interesting aerodynamics will be seen through that cavity.

Two different views of the payload model can be seen in Figures 8.6 and 8.7. The cavity in the payload was retained in the model. The model was created to convey
the general shape of the prototype, but specifics, such as the crossbars in Figure 8.5, were not necessary in analyzing the trajectory of general cargo payloads and how much they are influenced by the aerodynamics of the specific cargo aircraft. Again, in this problem, the focus falls on only the first 2–3 seconds of the payload drop. The parachute dynamics have been analyzed in other works [30,38].

Again, three-dimensional triangular surface meshes were created from the models.
And then the 3-D tetrahedral volume mesh was generated for the fluid dynamics solution using these surface meshes. This mesh consists of 332,498 nodes and 1,952,559 tetrahedral elements. It is important to note that the computational costs, here, are greater than that of the paratrooper simulation. This is due to the use of the full model of the aircraft whereas the paratrooper simulation utilizes a half model. The modeling software, 3D surface mesh generator, and automatic 3D mesh generator were the same used for the paratrooper simulation, all developed by Dr. Andrew Johnson [24].

8.2 FOI Setup

In this example of a payload drop, the speed of the cargo aircraft, the angle of attack of the aircraft, the boundary conditions for the problem, and the length of the aircraft are the same as that in Section 7.2. The same is true of the engine intake and exhaust flow velocities and the free-stream velocity. The only difference in this simulation is the separating object--cargo payload--and the model of the aircraft--a full model with a upper cargo bay doors opening upwards and inwards and lower cargo bay doors opening downwards and outwards, Figure 8.4.
8.3 FOI Simulation

The payload drop simulation was computed, and Figure 8.8 displays the pressure distribution for this simulation. Notice the high pressure inside the cavity of the payload. This will affect the pressure the payload feels. Figures 8.9 show the stream-ribbons that travel around the cargo room and the back of the plane. Notice that some ribbons actually enter the cargo room. A side view is shown in Figure 8.10.

![Pressure Distribution](image)

Figure 8.8. Pressure Distribution on Cargo Payload and Aircraft.

For this simulation, the payload slides along a frictionless cargo floor before tipping over the edge and falling down to the ground below. The code utilized for this thesis does not have contact capabilities, so the sliding motion of the cargo payload had to be prescribed. So to prescribe this non-contacting motion, some tiny bit of air between the bottom of the payload and the cargo floor was allowed. Figure 8.11 displays a surface mesh of the payload in the pre-tipping stage. Figure 8.12 demonstrates the same, now allowing the air gap between the floor and the bottom of the payload to be viewable. A trajectory for the payload drop was simulated using the described models and specifications. This trajectory is shown in Figure 8.13.

Here, as with the paratrooper jump, the validation of this simulated trajectory is a key issue. Is this a possible real trajectory? Is it believable? In the case of the payload drop, there were no videotapes given to me to allow such a loose comparison. This is even less information than what was available for the paratrooper jumps. So
my comparison is now made only between my simulated trajectory and an analytical trajectory.

The analytical trajectory was calculated using an estimated coefficient of drag—1.5. Using Equation 7.1, again, and the same values for the variables in the equation, the drag force and trajectory were calculated analytically using the free-stream value, as in Section 7.3.

Figure 8.14 compares the analytical trajectory to the numerical one. It can be observed that the trajectory produced from the numerical simulation is a possible, viable trajectory when compared to the analytical trajectory. The slope, measured using the starting and end points, of the analytical trajectory is -2.347 and that of the numerical or computational one is -2.324. They are very close. And there seems to be no drag problem here when compared to the analytical solution because the numerical trajectory is swept back more!

Still, some may feel that the cargo, if a real, physical trajectory were obtained, would and should be swept back even more. I believe that the cargo trajectory is gravity-dominated. In other words, any effects of a thick numerical boundary layer
will not be as prominent as they were in the paratrooper simulation. This is because the thick numerical boundary layer effects the calculation of drag. Since gravity dominates here, a slip formulation should have a smaller effect than it did in the paratrooper simulation. I can still apply the slip method from Chapter 6 to increase the drag felt by the cargo and view the improvement on the trajectory.

8.3.1 Slip Simulation

Now I attempt to simulate the cargo payload falling from the cargo bay of the aircraft with newly applied slip conditions. The boundary conditions are the same as stated in Section 8.2 with the one change applying to the surface of the aircraft. It is now given a slip condition allowing the fluid particles to flow along the surface of the aircraft, thereby eliminating the boundary layer and the low drag problem. Figures 8.15 and 8.16 show this new trajectory.

And now I add this new trajectory in my comparison of trajectories. Figure 8.17 clearly compares all trajectories and validates the numerical slip results. Using
the slip approximation has provided a trajectory even more swept back. And in this case, there is definitely no boundary layer thickness problem. Again, the slip code formulation has proven a new possible, viable, and real trajectory for the high drag experience of a cargo payload. And as hypothesized, the improvement to the trajectory using a slip formulation was less pronounced here than it was in Section 7.3 for the paratrooper simulation which is drag-dominated.

8.3.2 FOIST Simulation

Lastly, in this chapter, I wish to examine the effects of the new FOIST method
on this complex 3D simulation of the falling cargo payload, a new application. As I did in the paratrooper simulation, I wish to determine if FOIST approximates the trajectory well enough that the cost savings are justified. The falling payload simply provides another example to test the capabilities of FOIST.

First, a subcomputational domain containing only the payload was modeled. This is displayed in Figures 8.18 and 8.19. Then a main flow was simulated around the full-modeled cargo aircraft without the payload assuming, according to FOIST philosophy, that the influence of the payload on the main flow field is negligible.

Using the same specifications and conditions outlined in Section 8.2, projecting from the main flow field, the FOIST subcomputation around the payload was simulated, Figure 8.20.

The results are not surprising and follow the approximations FOIST made for the jumping paratrooper simulation. For the same reasons, the drag experience of the cargo payload using the FOIST formulation is less, as expected, due to the aforementioned approximations with FOIST (Chapter 5). And, in this case, as well, the approximation is satisfactory in presenting a general estimate of the shape and size of the payload trajectory. And, as stated before in Section 7.3, such an approximation can still be helpful for prognostic analyses for drop location.
Figure 8.13. Trajectory Snapshots of Payload Drop Simulation.
Figure 8.14. Cargo Payload Trajectory Comparison: Numerical vs Analytical.

Figure 8.15. Payload Trajectory with Slip Conditions on Aircraft.
Figure 8.16. Last Snapshot of Payload Trajectory with Slip Conditions on Aircraft.

Figure 8.17. Cargo Payload Trajectory Comparison: Numerical vs Analytical vs Slip.
Figure 8.18. Model of FOIST Subcomputational Domain Containing Payload.

Figure 8.19. Side View of Colored Payload in Subcomputational Domain for FOIST.
Figure 8.20. Cargo Payload Trajectory Comparison: Numerical vs Analytical vs Slip vs FOIST.
Chapter 9

Conclusions

The aerodynamic interaction of separating objects from larger bodies is an important problem. And it occurs in situations were paratroopers are jumping from cargo aircraft or when any objects, or payloads, are dropped from cargo bays of such aircraft. These FOI problems create a difficult challenge due to their complex 3-D geometries and the complex unsteady 3-D nature of the problems. In addition, a

Figure 9.1. Paratroopers Exiting Cargo Aircraft.
large system of equations governing the fluid flow is to be solved at each time step.

The solution of such problems consists of three segments: the fluid dynamics simulation, the mesh moving calculations, and the object or particle dynamics. The fluid dynamics and the object dynamics are coupled together. And when the object is allowed to deform, as well as translate and rotate, the resulting interactions are termed Fluid-Structure Interactions. Here, the separating objects were treated rigidly and only permitted to undergo translation and rotation.

The applications and special emphases of this thesis are paratrooper–aircraft separation problems, or jumping paratroopers, with a minor focus on payload drops from cargo aircraft. With all the movement of these rigid bodies, there was a need to take into account this movement and resultant time-dependent spatial domain. Solutions for the fluid dynamics were obtained through computation with the DSD/SST procedure. The DSD/SST procedure automatically handles deforming boundaries by integrating over the space-time domain. Due to the motion of the separating object geometry changes to the fluid mesh were handled using an automatic mesh moving scheme. This was accomplished by treating the mesh domain as a pseudo-elastic solid and assigning prescribed displacements at the boundary of the mesh that surrounds the moving object. Those displacements naturally came from the movement of the object.

Methods to update the mesh and handle mesh moving for moving objects were presented. Different mesh stiffening tactics were explored to delay and reduce the amount of remeshing and projection and the resultantly introduced errors. Aspect-ratio based mesh stiffening was found to be the best stiffening tactic. Also a new divergence-free projection solver was implemented to avoid pressure spikes, after remeshes, caused by errors from integration during projection. This projection solver was found only to reduce the parts of the spikes resulting from this error, but to leave the natural part of the pressure spikes resulting from improved accuracy in newly made Delauney mesh with an average aspect ratio around 1.0.

Then a mesh moving alternative was proposed: FOIST—Fluid Object Interactions Subcomputation Technique. FOIST is a good approximator of a general trajectory
for separating objects. In general, when compared to the same simulation using mesh moving techniques, objects experienced less drag in FOIST computations. Therefore, more work needs to be done in improving this alternative, so that I may ascertain the specifics of the predicted trajectory.

Lastly introduced were methods for the reduction of numerically thick boundary layers. A boundary layer program was implemented. This program accompanies Andrew Johnson’s 3-D mesh generator ([39]) and creates layers of tetrahedral boundaries evenly or with a growth factor of two. The importance of this is great as there are no 3-D boundary layer mesh generators for arbitrary shapes and geometries that use tetrahedral elements in the boundary. However, the effectiveness of the tetrahedral boundary layers depends on the problem. As noticed, it did little to reduce the boundary layer on the cargo aircraft partly due to its angle of attack of eight degrees and mostly due to the widening diameter of the fuselage. In these cases, more layers are needed, but that can be beyond computational resources. So a slip formulation was implemented, as well, for arbitrary geometries. And this formulation removed the boundary layer as an approximation of a very thin layer. And this increased the drag felt for separating objects. The paratrooper, for example, was clearly swept back more with a larger drag force.

Lastly the applications of a jumping paratrooper and dropped payload were examined. The simulation of the falling cargo payload was found to be gravity dominated due to the large weight. So the drag error did not affect the mesh moving simulation much and the mesh moving simulation approximated analytical results very well. In fact, the mesh moving simulation was swept back more than the analytical solution. And the slip formulation did not sweep the trajectory back as much as it did for the case of the jumping paratrooper whose trajectory is drag dominated. For the paratrooper simulation, the mesh moving simulation produced a trajectory that could have been a possible trajectory due to the randomness of the trajectories of the actual paratrooper jumps. It produced a trajectory close to the real, physical trajectory of a paratrooper least swept back. So the slip formulation helped greatly and allowed the the modeled paratrooper to be swept back even more than the videotaped trajectory
of the real paratrooper who was most swept back in the videotaped jumps. FOIST simulations for both the paratrooper jumps and cargo drops approximated the general shape of the mesh moving trajectories. But the moving objects in FOIST experienced much less drag force than the mesh moving simulations. FOIST simulations for both the paratrooper jumps and cargo drops approximated the general shape of the mesh moving trajectories. But the moving objects in FOIST experienced much less drag force than the mesh moving simulations. So more work must be done to remove some of the approximations FOIST introduces in order to reach a closer trajectory, though FOIST can be used as a general indicator.
In order to further enhance the FOI simulations of separating objects and techniques that are presented in this thesis, a number of future research directions could be explored:

- A model to handle contact for rigid and deformable bodies could be added to allow greater flexibility in simulations and to increase the realism of the models and thereby the results.

- When the quality of the existing mesh becomes too poor, a method to "remesh" while retaining all nodes and element connectivity would aid my projection step. In this case, the pressure spikes should be totally eradicated, since that which was not removed by the divergence-free projection solver is due to the change in element connectivity and new nodal distribution.

- FOIST has much room for improvement. Two ways to increase its accuracy and reduce the approximations introduced include the following:
  
  - For each time step of the subcomputation, I could iterate between the projected boundary node values and the resulting fluid solution and Newtonian forces and displacement. It would be interesting to note the result of increasing the number of iterations beyond one iteration before moving to the next time step.
  
  - The introduction of nodal boundary conditions would greatly aid in the approximation of the entire boundary as inflow, outflow, or slip boundary conditions. This could allow the true representation of some boundaries as partially inflow and partially outflow and/or partially slip.

- In order to accurately resolve the numerically thick boundary layer, I must attempt to add the boundary layer capability to my automatic 3-D mesh generator. The tetrahedral boundary layer resulting from the mesh generator itself should be superior to that created on an existing mesh since an automatic mesh generator can create a tetrahedral boundary layer of elements first before filling in the rest of the mesh.
Appendix A

Appendix

Figure A.1. Trajectory of Ball-Shear Flow: FreeStream vs Transverse Directions.
Figure A.2. Trajectory of Ball in Shear Flow: Gravity-Direction vs Transverse Direction.

Figure A.3. Red - Freestream Value vs Blue < 100% Freestream Value.
Figure A.4. Dichotomous Velocity Picture: Red $\geq$ 90% Freestream Value, Blue $<$ 90% Freestream Value.

Figure A.5. Dichotomous Velocity Picture: Red $\geq$ 90% Freestream Value, Blue $<$ 90% Freestream Value.
Figure A.6. Dichotomous Velocity Picture: Red $\geq$ 85% Freestream Value, Blue < 85% Freestream Value.

Figure A.7. Dichotomous Velocity Picture: Red $\geq$ 80% Freestream Value, Blue < 80% Freestream Value.
Figure A.8. Dichotomous Velocity Picture: Red ≥ 75% Freestream Value, Blue < 75% Freestream Value.

Force on Paratrooper in Freestream Direction

Force on Paratrooper in Transverse (Wingspan) Direction

Figure A.9. Effect of Inertia Scalar (Left) vs Inertia Matrix (Right).
Bibliography


