RICE UNIVERSITY

Special Methods for Fluid–Object Interactions and Space–Time Computations

by

Ryan M. Keedy

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

MASTER OF SCIENCE

Approved, Thesis Committee:

Tayfun E. Tezduyar
Professor, Chair, Advisor
Mechanical Engineering and Mat. Science

J. Ed Akin
Professor
Mechanical Engineering and Mat. Science

Marcia K. O’Malley
Assistant Professor
Mechanical Engineering and Mat. Science

Houston, Texas
January, 2004
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.
Abstract

Special Methods for Fluid–Object
Interactions and Space–Time Computations

by

Ryan M. Keedy

Simulations of complex fluid–object interactions problems in aeronautics demand robust and sophisticated numerical techniques. The proposed B-FOIST is an efficient library-lookup method for predicting the response of an object to a dominant, arbitrary flow field. B-FOIST predicts the trajectory of an object without the need for flow subcomputations, mesh-moving or remeshing. Subsequently, the path of the object can be calculated more quickly and efficiently than traditional mesh-moving methods while producing comparable results.

Implementation of the proposed SSTF formulation can improve the efficiency of traditional space–time finite element computations. Many repetitive, unnecessary calculations can be eliminated by reformulating shape function derivatives and restructuring the element-level matrix/vector calculations. There is the potential for large computational savings depending on the previous structure of the computations.
Acknowledgements

I am very thankful to Dr. Tayfun Tezduyar for giving me the opportunity, early in my Rice career, to be involved in cutting-edge research with the Team for Advanced Flow Simulation and Modeling. T*AFSM has opened countless doors for me that would have otherwise remained forever locked.

Thanks to my other committee members, Dr. Ed Akin and Dr. Marcia O’Malley, for willingly giving up their time to sit on my defense.

I would like to thank Victor Udoewa for taking me under his wing when I had no idea which way was up. His patience with my ignorance gave me the encouragement I needed to stick with it.

Thanks to Sunil Sathe for turning his very own 2-D code over to me so that I could butcher it up any number of ways. He was very kind to share his knowledge and experience with me throughout my time here.

I am thankful to Vinod Kumar for his inspiring pictures, Masayoshi Senga for his generosity with chocolate, Mehmet Ungor for his light-hearted humor, Tomas Soltys for his optimism, and Medhat El-Nahas for his unique perspectives.

Thanks also to all of my teachers throughout the years. I would not be where I am today without their often unappreciated hard work. A special thanks Mr. Scotty Johnson, without whom I would never have made it to or at Rice.

None of this could be possible of course without the love and support of my friends and family through the good times and the bad. My parents were, of course,
instrumental to my personal development as both a student and a human being, for which I will be forever indebted to them.

Finally, I would like to thank the Army Natick Soldier Center, National Science Foundation, and NASA JSC, whose financial support made this research possible.
Contents

Abstract ii

Acknowledgments iii

List of Figures viii

List of Tables x

1 Introduction 1

2 Governing Equations 5
   2.1 Balance Equations ........................................... 5
   2.2 Constitutive Relationship ................................... 6
   2.3 Boundary Conditions ........................................... 6
   2.4 Particle Dynamics Equations ................................... 6
   2.4.1 Forces on Objects ........................................... 7
   2.4.2 Rigid-Body Motion ........................................... 8

3 DSD/SST Finite Element Formulation 9

4 Beyond FOIST 12
   4.1 Introduction to FOIST ........................................... 12
   4.2 Introduction to B-FOIST ........................................... 13
4.3 Implementation of B-FOIST ........................................ 14
  4.3.1 2D Implementation ........................................... 17
4.4 Problem Considerations ........................................... 18
  4.4.1 Steady vs. Unsteady Flow .................................. 19
  4.4.2 Multiple Objects ........................................... 20
  4.4.3 Accounting for an Object’s Angular Velocity ............ 21
  4.4.4 Uniformity of Fluid Velocity ............................. 22
4.5 Library Considerations .......................................... 23
  4.5.1 Composition of Library ................................... 23
  4.5.2 Predictable Orientations .................................. 25
  4.5.3 Planes of Symmetry ....................................... 26
4.6 2D Ellipse Simulation ........................................... 28
  4.6.1 FOI Problem Characterization ............................ 28
  4.6.2 B-FOIST Problem Characterization ...................... 31
  4.6.3 Numerical Results ....................................... 36
4.7 Future Work ................................................... 55

5 Special Space–Time Formulation ................................. 56
  5.1 Introduction .................................................. 56
  5.2 Shape Function Derivatives .................................. 57
  5.3 Element Vector Definition ................................... 62
    5.3.1 Derivatives with Respect to Time ....................... 63
  5.4 SSTF Implementation ........................................ 65
    5.4.1 Implementing Shape Functions Derivatives ............ 66
    5.4.2 Implementing Element Vector Definition ............... 67
  5.5 SSTF Results ................................................ 70
  5.6 Future Work ................................................ 76
6 Conclusions

6.1 Beyond FOIST .................................................. 77
6.2 Special Space-Time Formulation .............................. 78

A SSTF Element-Level Matrix

A.1 Element Matrix Definition .................................... 79
   A.1.1 Derivatives with Respect to Time ...................... 80
A.2 SSTF Implementation ........................................ 81

Bibliography ....................................................... 83
## List of Figures

3.1 Space-time slab concept drawing ............................................. 10

4.1 Definition of flow angles in 2D .............................................. 18

4.2 2D mesh diagram ................................................................. 29

4.3 Close-up view of 2D ellipse mesh ......................................... 30

4.4 Quadratic prediction of force ............................................... 32

4.5 Body force predictions for 2D ellipse ..................................... 34

4.6 Moment history for rotated 2D ellipse, $\omega = 1.0$ ................... 35

4.7 Force histories for translating 2D ellipses ($m = 2500$) ............... 38

4.8 Force histories for translating 2D ellipses ($m = 10000$) ............. 38

4.9 Displacement histories for translating 2D ellipses ($m = 2500$) ...... 39

4.10 Displacement histories for translating 2D ellipses ($m = 10000$) .... 39

4.11 Moment histories for rotating 2D ellipses ($I = 1000$) ............... 42

4.12 Moment histories for rotating 2D ellipses ($I = 4000$) ............... 42

4.13 Angular velocity histories for rotating 2D ellipses ($I = 1000$) .... 43

4.14 Angular velocity histories for rotating 2D ellipses ($I = 4000$) .... 43

4.15 Angular displacement histories for rotating 2D ellipses ($I = 1000$) 44

4.16 Angular displacement histories for rotating 2D ellipses ($I = 4000$) 44

4.17 Moment histories for damped/undamped rotating 2D ellipses ($I = 4000$) 45

4.18 Angular displacement histories for damped/undamped rotating 2D ellipses ($I = 4000$) .................................................. 45
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.19</td>
<td>Force histories for rotating 2D ellipses ($I = 4000$)</td>
<td>46</td>
</tr>
<tr>
<td>4.20</td>
<td>Force histories for free 2D ellipses ($m = 2500, I = 1000$)</td>
<td>48</td>
</tr>
<tr>
<td>4.21</td>
<td>Force histories for free 2D ellipses ($m = 10000, I = 4000$)</td>
<td>49</td>
</tr>
<tr>
<td>4.22</td>
<td>Velocity histories for free 2D ellipses ($m = 2500, I = 1000$)</td>
<td>50</td>
</tr>
<tr>
<td>4.23</td>
<td>Velocity histories for free 2D ellipses ($m = 10000, I = 4000$)</td>
<td>51</td>
</tr>
<tr>
<td>4.24</td>
<td>Displacement histories for free 2D ellipses ($m = 2500, I = 1000$)</td>
<td>52</td>
</tr>
<tr>
<td>4.25</td>
<td>Displacement histories for free 2D ellipses ($m = 10000, I = 4000$)</td>
<td>53</td>
</tr>
<tr>
<td>4.26</td>
<td>Displacement histories for damped/undamped free 2D ellipses ($m = 10000, I = 4000$)</td>
<td>54</td>
</tr>
<tr>
<td>5.1</td>
<td>Space–time element concept drawing</td>
<td>58</td>
</tr>
<tr>
<td>5.2</td>
<td>Timings of different sections of the subroutine calculating element-level vectors</td>
<td>72</td>
</tr>
<tr>
<td>5.3</td>
<td>Timings of different sections of the subroutine calculating element-level vectors (SSTF)</td>
<td>72</td>
</tr>
<tr>
<td>5.4</td>
<td>Timings of portions of vector-based 2D flow solver, $nts = 1$</td>
<td>74</td>
</tr>
<tr>
<td>5.5</td>
<td>Timings of portions of vector-based 2D flow solver, $nts = 1$ (SSTF)</td>
<td>74</td>
</tr>
<tr>
<td>5.6</td>
<td>Timings of portions of vector-based 2D flow solver, $nts = 5$</td>
<td>75</td>
</tr>
<tr>
<td>5.7</td>
<td>Timings of portions of vector-based 2D flow solver, $nts = 5$ (SSTF)</td>
<td>75</td>
</tr>
</tbody>
</table>
List of Tables

4.1 Library of force coefficients for 2D ellipse ....................... 33
4.2 Library of moment coefficients for 2D ellipse ....................... 33

5.1 Influence of SSTF on vector-based code execution time ............. 73
5.2 SSTF code duration as a percentage of original program ............. 73
Chapter 1

Introduction

Finite element methods have been successfully used to solve a wide variety of fluid dynamics problems. Each type of problem demands an approach that balances accuracy, stability, and computational cost in the most effective way. Often, there are several approaches to a given problem, each with their advantages and limitations. Depending on the demands of a problem, different strengths will be emphasized while deficiencies are overlooked. For instance, if only a rough estimation of a solution is needed, the desire for accuracy may take a back seat to reducing computational cost. On the other hand, if computational resources are vast, the cost of a computationally intense solution may be acceptable for the accuracy that can be gained.

The class of problems involving moving boundaries can be one of the most costly for finite element methods. In addition to the normal fluid computations needed for a fluid–object interaction simulation, a separate system of equations determining the motion of the mesh nodes must be solved if the mesh is unstructured. Remeshing will be necessary when the finite element mesh becomes too distorted to yield accurate results. This will consume more computational time, and it will also introduce errors during the projection of the old solution onto the new nodes. Depending on the geometry and the duration of the problem, this undesirable remeshing procedure
may need to be executed dozens of times.

Certain types of prescribed (relative linear or rotational) motion can be made more efficient by implementing special-purpose mesh-moving techniques such as the Shear–Slip Mesh Update Method (SSMUM) [14, 2, 3]. Similarly, problems involving an object traveling along an unknown trajectory demand unique methods of their own. Tomaro, Witzeman and Strang have previously predicted the trajectory of a missile being ejected from an F-18C without the use of mesh motion [18]. The method they employed uses a decay function to interpolate between free stream and carriage loads on the ordinance. However, this approach fails to thoroughly examine the aerodynamics found in the transition region.

We would like an accelerated technique to accurately solve problems involving moving boundaries with unknown trajectories, while incorporating the entire domain’s flow field into our formulation. Tezduyar [10, 12] proposed two related techniques for dealing with such a problem. Both operate under the assumption that the nature of the problem allows us to consider only one-way coupling from the greater flow field to the object. The two techniques will only work for rigid body motion and would be inappropriate for problems with deforming structures (i.e. parachute modeling).

The Fluid–Object Interactions Subcomputation Technique (FOIST) is the more computationally intense of the two proposed techniques. With the assumption of one-way coupling, we will disregard an object’s effect on the main flow field. This allows us to calculate the entire domain’s flow field in a fixed mesh without the moving objects. The one-way coupling will also grant us the freedom to conduct fluid subcomputations in smaller, fixed domains containing the objects. Boundary conditions for these subdomains can be harvested from the main flow field based on the predicted location and orientation of the objects in the flow field. Subcomputations can be carried out for each object to determine the instantaneous forces and mo-
ments which are then used to compute the objects' incremental displacements. The boundary conditions are again retrieved from the main domain at the next time-step at each object's recalculated position and orientation. Done repetitively, an object's trajectory and orientation over a period of time can be computed without the need for mesh-moving.

There are simplifications that can be made to FOIST which will result in a new method, Beyond FOIST (B-FOIST). We will eliminate the subcomputations that are carried out at each time step with boundary conditions retrieved from the larger domain. Instead, a library of forces and moments is compiled by separately evaluating each object under various orientations and uniform inflow velocities. Then the object's orientation, as well as the fluid velocity at its location in the larger domain are used to interpolate among the library entries and predict the force and moment felt by the object at that instant in time. The force and moment will be used to predict the object's acceleration, and eventually its new position and orientation in the larger domain an infinitesimal amount of time later, where the force and moment can be re-interpolated. Through this iterative process, the trajectory of each object can be calculated extremely quickly.

There are many types of problems that are either inappropriate for B-FOIST or require supplemental simulations. In these instances, a fully coupled fluid–object interactions formulation is needed. The Special Space–Time Formulation (SSTF) can be implemented to improve the efficiency of typical space–time finite element computations.

Most space–time meshes are structured in time, and even as spatial mesh motion occurs, the time slabs remain parallel. This knowledge can be used to eliminate repetitive calculations in two ways. It is important to note that the calculations that are revised by SSTF are a significant portion of the total operations carried out by a fluid–objects interaction program.
First, the derivatives of the shape functions must be calculated for each element in the mesh, depending on its shape, size, and orientation. These derivative calculations can be simplified, knowing that the time slabs are always parallel.

Second, the element-level matrices and/or vectors are normally integrated over all $2 \times n_{en}$ shape functions in space and time. By restructuring the integration procedure, the integration can be reduced to a summation over only the $n_{en}$ spatial shape functions with minor post-processing. Depending on the structure and composition of the current formulation, there may be the potential for significant computational savings.
Chapter 2

Governing Equations

2.1 Balance Equations

Let $\Omega_t \subset \mathbb{R}^{n+1}$ be the spatial fluid mechanics domain with boundary $\Gamma_t$ at time $t \in (0, T)$, where the subscript $t$ indicates the time-dependence of the spatial domain. The Navier–Stokes equations of incompressible flows can be written over $\Omega_t$ and $\forall t \in (0, T)$ as

\[
\rho \left( \frac{\partial u}{\partial t} + u \cdot \nabla u - f \right) - \nabla \cdot \sigma = 0, \quad (2.1)
\]

\[
\nabla \cdot u = 0, \quad (2.2)
\]

where $\rho$, $u$ and $f$ are the density, velocity and the external force, respectively.

These Navier–Stokes equations, along with $\sigma$ as defined in Section 2.2, together with the proper boundary and initial conditions compose, our governing system of equations.
2.2 Constitutive Relationship

The stress tensor $\sigma$ is defined as

$$\sigma(p, u) = -p I + 2\mu \varepsilon(u).$$  \hspace{1cm} (2.3)

Here $p$ is the pressure, $I$ is the identity tensor, $\mu = \rho \nu$ is the viscosity, $\nu$ is the kinematic viscosity, and $\varepsilon(u)$ is the strain-rate tensor:

$$\varepsilon(u) = \frac{1}{2} \left( (\nabla u) + (\nabla u)^T \right).$$  \hspace{1cm} (2.4)

2.3 Boundary Conditions

The essential and natural boundary conditions for equation (2.1) are represented as

$$u = g \text{ on } (\Gamma_t)_g, \quad n \cdot \sigma = h \text{ on } (\Gamma_t)_h,$$  \hspace{1cm} (2.5)

where $(\Gamma_t)_g$ and $(\Gamma_t)_h$ are complementary subsets of the boundary $\Gamma_t$, $n$ is the unit normal vector, and $g$ and $h$ are given functions. A divergence-free velocity field $u_0(x)$ is specified as the initial condition.

2.4 Particle Dynamics Equations

Many fluid flow problems involve additional equations relating the fluid and the boundaries in the domain. Of particular concern are the equations that determine the forces imposed by the fluid on objects in the domain, and the equations of motion governing the particle dynamics.
2.4.1 Forces on Objects

In many situations, it is important to recover the forces imposed by the fluid on an object from the fluid flow calculation. These forces may be used to determine the loads and stresses on an object of concern, they may dictate the performance characteristics of an aerodynamic object, or they may be used to predict the dynamics of an object allowed to move freely through the fluid flow.

The surface force per unit area at a point on an object with surface \( \Gamma \) due to the fluid around it is defined by

\[
\mathbf{\sigma} \cdot \mathbf{n} \quad \text{on} \quad \Gamma, \tag{2.6}
\]

where \( \mathbf{\sigma} \) is the stress tensor introduced in Section 2.2, and \( \mathbf{n} \) is the unit vector normal to the surface \( \Gamma \). Expanding on the definition presented in Section 2.2, \( \mathbf{\sigma} \) can be rewritten as

\[
\mathbf{\sigma} = \begin{bmatrix}
-\rho + 2\mu \frac{\partial u_1}{\partial x_1} & \mu \left( \frac{\partial u_2}{\partial x_1} + \frac{\partial u_1}{\partial x_2} \right) & \mu \left( \frac{\partial u_1}{\partial x_1} + \frac{\partial u_1}{\partial x_2} \right) \\
\mu \left( \frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1} \right) & -\rho + 2\mu \frac{\partial u_2}{\partial x_2} & \mu \left( \frac{\partial u_2}{\partial x_2} + \frac{\partial u_2}{\partial x_3} \right) \\
\mu \left( \frac{\partial u_1}{\partial x_3} + \frac{\partial u_3}{\partial x_1} \right) & \mu \left( \frac{\partial u_2}{\partial x_3} + \frac{\partial u_3}{\partial x_2} \right) & -\rho + 2\mu \frac{\partial u_2}{\partial x_2}
\end{bmatrix}, \tag{2.7}
\]

where \( u_1, u_2 \) and \( u_3 \) are the \( x_1, x_2 \) and \( x_3 \) components of the velocity vector. Now that we have defined the stress, we can define the components of the force per unit area:

\[
(\mathbf{\sigma} \cdot \mathbf{n})_1 = \left( 2\mu \frac{\partial u_1}{\partial x_1} - \rho \right) n_1 + \mu \left( \frac{\partial u_2}{\partial x_1} + \frac{\partial u_1}{\partial x_2} \right) n_2 + \mu \left( \frac{\partial u_3}{\partial x_1} + \frac{\partial u_1}{\partial x_3} \right) n_3, \tag{2.8}
\]

\[
(\mathbf{\sigma} \cdot \mathbf{n})_2 = \left( 2\mu \frac{\partial u_2}{\partial x_2} - \rho \right) n_2 + \mu \left( \frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1} \right) n_1 + \mu \left( \frac{\partial u_3}{\partial x_2} + \frac{\partial u_2}{\partial x_3} \right) n_3, \tag{2.9}
\]

\[
(\mathbf{\sigma} \cdot \mathbf{n})_3 = \left( 2\mu \frac{\partial u_3}{\partial x_3} - \rho \right) n_3 + \mu \left( \frac{\partial u_1}{\partial x_3} + \frac{\partial u_3}{\partial x_1} \right) n_1 + \mu \left( \frac{\partial u_2}{\partial x_3} + \frac{\partial u_3}{\partial x_2} \right) n_2. \tag{2.10}
\]

The total force on an object in the flow field can be obtained by integrating over
the entire surface of the object. Therefore, the equation for the total force is:

\[ F = \int_{\Gamma} \sigma \cdot n d\Gamma. \]  

(2.11)

Similarly, the moment vector about a point \( X_{O} \) is given as:

\[ M_{X_{O}} = \int_{\Gamma} (x - x_{O}) \times (\sigma \cdot n) d\Gamma. \]  

(2.12)

### 2.4.2 Rigid-Body Motion

Some fluid flow problems involve unfettered objects, free to move throughout the flow. In these problems, the objects will translate and rotate in response to the fluid forces and moments upon them. This object movement will in turn affect the fluid flow, which will affect the object, and so on. The equations for the forces on the particle were defined in the previous subsection, and these forces govern the motion of the objects as follows:

\[ \mathbf{F} = \mathbf{M} \dot{\mathbf{V}}, \]  

(2.13)

where \( \mathbf{F} \) is the composite force/moment vector, \( \mathbf{M} \) is the composite mass/moment of inertia matrix, and \( \dot{\mathbf{V}} \) is the time derivative of the composite linear/angular velocity vector. They can be written as

\[
\mathbf{F} = \begin{pmatrix} F_1 \\ F_2 \\ F_3 \\ M_1 \\ M_2 \\ M_3 \end{pmatrix}, \quad \mathbf{M} = \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{11} & I_{12} & I_{13} \\ 0 & 0 & 0 & I_{21} & I_{22} & I_{23} \\ 0 & 0 & 0 & I_{31} & I_{32} & I_{33} \end{bmatrix}, \quad \mathbf{V} = \begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix}. \]  

(2.14)
Chapter 3

DSD/SST Finite Element Formulation

The Deforming-Spatial-Domain/Stabilized-Space-Time (DSD/SST) formulation is an interface-tracking method, and was first introduced in [9, 15, 16]. The finite element formulation of the governing equations is written over a sequence of $N$ space-time slabs $Q_n$, where $Q_n$ is the slice of the space-time domain between the time levels $t_n$ and $t_{n+1}$ (see Figure 3.1).

At each time step, the integrations involved in the finite element formulation are performed over $Q_n$. The space-time finite element interpolation functions are continuous within a space-time slab, but discontinuous from one space-time slab to another. The notation $(\cdot)_n^-$ and $(\cdot)_n^+$ denotes the function values at $t_n$ as approached from below and above. Each $Q_n$ is decomposed into elements $Q_{n}^{e}$, where $e = 1, 2, \ldots, (n_{el})_{n}$. The subscript $n$ used with $n_{el}$ is for the general case in which the number of space-time elements may change from one space-time slab to another. The essential and natural boundary conditions are enforced over $(P_n)_g$ and $(P_n)_h$, the complementary subsets of the lateral boundary of the space-time slab. The finite element trial function spaces $(S^h_{U})_n$ for velocity and $(S^h_{P})_n$ for pressure, and the test function spaces $(V^h_{U})_n$ and $(V^h_{P})_n$...
Figure 3.1: Drawing of a typical space–time slab

\( (S_p^h)_n \) are defined by using, over \( Q_n \), first-order polynomials in both space and time.

The DSD/SST formulation is written as follows: given \( (u^h)_n \), find \( u^h \in (S_u^h)_n \) and \( p^h \in (S_p^h)_n \) such that \( \forall w^h \in (V_u^h)_n \) and \( q^h \in (V_p^h)_n \):

\[
\int_{Q_n} w^h \cdot \rho \left( \frac{\partial u^h}{\partial t} + u^h \cdot \nabla u^h - f^h \right) dQ + \int_{Q_n} \varepsilon(w^h) : \sigma(p^h, u^h) dQ \\
- \int_{(P_n)_h} w^h \cdot h^h dP + \int_{Q_n} q^h \nabla \cdot u^h dQ + \int_{\Omega_n} (w^h)_n^+ \cdot \rho \left( (u^h)_n^+ - (u^h)_n^- \right) d\Omega \\
+ \sum_{c=1}^{n_{el}^n} \int_{Q_c^h} \left[ \tau_{SUPG} \rho \left( \frac{\partial w^h}{\partial t} + u^h \cdot \nabla w^h \right) + \tau_{PSPG} \nabla q^h \right] \cdot \left[ L(p^h, u^h) - \rho f^h \right] dQ \\
+ \sum_{c=1}^{n_{el}^n} \int_{Q_c^h} \nu_{LSIC} \nabla \cdot w^h \rho \nabla \cdot u^h dQ = 0,
\] (3.1)

where

\[
L(q^h, w^h) = \rho \left( \frac{\partial w^h}{\partial t} + u^h \cdot \nabla w^h \right) - \nabla \cdot \sigma(q^h, w^h).
\] (3.2)

Here \( \tau_{SUPG} \), \( \tau_{PSPG} \) and \( \nu_{LSIC} \) are the SUPG, PSPG and LSIC (least-squares on incompressibility constraint) stabilization parameters. For ways of calculating \( \tau_{SUPG} \), \( \tau_{PSPG} \) and \( \nu_{LSIC} \), see [17, 11, 13]. This formulation is applied to all space–time slabs.
$Q_0, Q_1, Q_2, \ldots, Q_{N-1}$, starting with $(u^b)^0_0 = u_0$. 
Chapter 4

Beyond FOIST

4.1 Introduction to FOIST

The Fluid–Object Interactions Subcomputation Technique (FOIST) is an approximation of a mesh-moving fluid–object interactions formulation [10, 12]. FOIST assumes that there exists no two-way fluid–object coupling, and it is only appropriate for problems for which this is a reasonable approximation. The coupling is assumed to be one-way, in that the fluid flow affects the accelerations of the objects, but the objects play no role in determining the behavior of the fluid. When we discuss “objects”, we will be referring to rigid bodies that are not fixed in the domain. There may, of course, exist other bodies in the problem, but these bodies must be stationary and will be considered boundaries of the fixed-mesh domain.

With the assumption of one-way coupling, we will disregard each object’s effect on the main flow field. This allows us to calculate the main flow field in a fixed mesh devoid of any moving objects. The flow in this larger domain may or may not be steady.

One-way coupling will also allow us to conduct fluid flow subcomputations in smaller, fixed-mesh domains containing the objects. Boundary conditions for these
subdomains are harvested from the main flow field based on the predicted locations, orientations, and velocities of the objects in the flow field. The inflow boundary conditions are nodally interpolated by super-positioning the subdomain on the larger domain at the relevant time step.

With the imposed boundary conditions, the subcomputations can be carried out for each object to determine the instantaneous forces and moment which are in turn used to compute each object’s incremental displacement. The boundary conditions are again retrieved from the main domain at the next time-step at each object’s recalculated position and orientation. Done repetitively, an object’s trajectory and orientation over a period of time can be computed. Note that neither mesh-moving nor remeshing is necessary at any point in the FOIST procedure.

4.2 Introduction to B-FOIST

Beyond FOIST (B-FOIST) is based on an extension of the simplifying assumptions that were made for FOIST. Again, we will operate under the assumption that our problem is adequately modeled by one-way coupling from the fluid to the objects. We will now take it one step further by eliminating most of the finite element computations from our formulation in order to cut computational cost.

Again, we must start by using the finite element method to calculate the fluid flow in our large domain, absent of any objects. However, before we attempt to predict the paths of the objects, we will compile a library of fluid forces, $F^\alpha_\beta$, and moments, $M^\alpha_\beta$, for each unique object. The libraries are obtained by time-averaging the forces and moments acting on each object due to uniform flow at different angles of incidence ($\theta_\beta$ in 2D, $e_\beta$ in 3D) and fluid speeds ($U^\alpha$). (Note: the force $F^\alpha_\beta$ is not necessarily in the $e_\beta$ direction.)

Each object’s orientation relative to the direction of fluid flow, as well as the fluid
speed at its location in the problem domain, are used to interpolate among the library entries and predict the force and moment felt by the object at that instant in time. The force and moment will be used to predict the object’s acceleration, and eventually its new position and orientation in the larger domain an infinitesimal amount of time later, where the force and moment can be re-interpolated. Through this iterative process, the trajectory of the object can be calculated extremely quickly.

4.3 Implementation of B-FOIST

The interpolation of forces and moments in B-FOIST actually involves two interpolations, the first being among library directions and the second among fluid speeds. In order to predict the direction of the forces and moments, we will find a linear combination of directions in our library, unit vectors \( \mathbf{e}_1, \mathbf{e}_2, \) and \( \mathbf{e}_3, \) that is equivalent to a unit vector in the direction of flow, \( \mathbf{s}: \)

\[
\mathbf{s} = S_1 \mathbf{e}_1 + S_2 \mathbf{e}_2 + S_3 \mathbf{e}_3, \tag{4.1}
\]

where \( S_\beta \) is the weight assigned to \( \mathbf{e}_\beta. \)

The first step is to find the unit vectors \( \mathbf{e}_1, \mathbf{e}_2, \) and \( \mathbf{e}_3 \) “closest” to \( \mathbf{s}. \) A unit vector \( \mathbf{e}_{i_1} \) is “closer” than \( \mathbf{e}_{i_2}, \) if \( \mathbf{e}_{i_1} \cdot \mathbf{s} > \mathbf{e}_{i_2} \cdot \mathbf{s}. \)

In order to determine \( S_\beta, \) we must solve the following system:

\[
(e_1 \cdot e_1)S_1 + (e_1 \cdot e_2)S_2 + (e_1 \cdot e_3)S_3 = e_1 \cdot s, \tag{4.2}
\]

\[
(e_2 \cdot e_1)S_1 + (e_2 \cdot e_2)S_2 + (e_2 \cdot e_3)S_3 = e_2 \cdot s, \tag{4.3}
\]

\[
(e_3 \cdot e_1)S_1 + (e_3 \cdot e_2)S_2 + (e_3 \cdot e_3)S_3 = e_3 \cdot s. \tag{4.4}
\]

Note that \((e_1 \cdot e_1) = (e_2 \cdot e_2) = (e_3 \cdot e_3) = 1.\)

We must also ensure that \( S_1, S_2, S_3 \geq 0. \) In order to properly interpolate among
our library vectors, we will first select \( e_1 \) such that \( e_1 \cdot s \) is maximum among \( \{ e_β \cdot s \} \). We must now check for the special case where \( e_1 \cdot s = 1 \). If this is the case, the vectors are colinear, and we know \( S_1 = 1 \), \( S_2 = 0 \), and \( S_3 = 0 \).

If the vectors \( e_1 \) and \( s \) are not colinear, then we will proceed to find \( e_2 \), where \( e_2 \cdot s \) is the second maximum among \( \{ e_β \cdot s \} \). Again, we must check our progress for special circumstances. If \( s \cdot (e_1 \times e_2) = 0 \) then \( s \), \( e_1 \), and \( e_2 \) are coplanar, in which case \( S_3 = 0 \). We must still solve \( S_1 \) and \( S_2 \) according to the following simplified system of equations:

\[
\begin{align*}
(e_1 \cdot e_1)S_1 + (e_1 \cdot e_2)S_2 &= e_1 \cdot s, \\
(e_2 \cdot e_1)S_1 + (e_2 \cdot e_2)S_2 &= e_1 \cdot s.
\end{align*}
\tag{4.5}
\tag{4.6}
\]

If neither special case exists, that is, our velocity vector \( s \) is not colinear or coplanar to either of the two closest vectors, then we must begin what may be an iterative procedure to find \( e_3 \). We would like to find \( e_3 \) so that the volume formed by \( e_1 \), \( e_2 \) and \( e_3 \) includes \( s \), and no other direction vector \( e_β \). In this way, \( s \) can be formed by a linear combination of \( e_1 \), \( e_2 \) and \( e_3 \). Having already found \( e_1 \) and \( e_2 \), our search will proceed as follows:

1. First, we will select the next “closest” library vector, i.e. \( (e_3 \cdot s) \) is the next maximum.

2. Solve equations (4.2), (4.3), and (4.4).

3. If \( S_1 > 0 \) and \( S_2 > 0 \) and \( S_3 \geq 0 \) then the search is finished.

If the search fails, eliminate the current \( e_3 \) from our search, change the condition \( S_2 > 0 \) to \( S_2 \geq 0 \), and re-do the search.

Once we have solved for \( S_1 \), \( S_2 \) and \( S_3 \), we must now interpolate among the fluid
speeds stored in our library. When interpolating the forces between our library's fluid speeds, we will follow a much simpler procedure, but we now have more options for our interpolation scheme. The most straightforward option is to linearly interpolate between the two fluid speeds closest to $U$, $U^1$ and $U^2$ (assume $U^1 \leq U \leq U^2$):

\begin{align*}
\nu_1 &= \frac{U^2 - U}{U^2 - U^1}, \\
\nu_2 &= \frac{U - U^1}{U^2 - U^1}.
\end{align*}

(4.7) (4.8)

However, for many situations, we know that the force and moment coefficients are a function of the square of the velocity ($F \propto (U)^2$, $F_2 = \frac{(U_2)^2}{(U_1)^2} F_1$). Therefore, we should also consider a quadratic interpolation scheme between library speeds. Let us redefine our interpolation coefficients to reflect our more sophisticated view of the relationship between force and speed. We will linearly interpolate between the quadratic predictions from $U^1$ and $U^2$:

\begin{align*}
\nu_1 &= \frac{(U)^2}{(U_1)^2} \frac{U^2 - U}{U^2 - U^1}, \\
\nu_2 &= \frac{(U)^2}{(U_2)^2} \frac{U - U^1}{U^2 - U^1}.
\end{align*}

(4.9) (4.10)

In either case, the interpolation of our force and moment with respect to direction and speed is simply

\begin{align*}
F_{avg} &= \nu_1 (S_1 F_1^1 + S_2 F_2^1 + S_3 F_3^1) + \nu_2 (S_1 F_1^2 + S_2 F_2^2 + S_3 F_3^2), \\
M_{avg} &= \nu_1 (S_1 M_1^1 + S_2 M_2^1 + S_3 M_3^1) + \nu_2 (S_1 M_1^2 + S_2 M_2^2 + S_3 M_3^2).
\end{align*}

(4.11) (4.12)

Note that the weights we assign to the different directions, $S_1$, $S_2$ and $S_3$ are not simply a function of how close they lie to the fluid velocity direction. Instead, they are weighted in order to take a linear combination of the three vectors to form $s$. In
a library with six orthogonal directions, we are guaranteed that the closest library direction will be assigned the largest weight, the second closest will have the second largest weight, etc (i.e. \( S_1 \geq S_2 \geq S_3 \)).

However, with more than six directions (that is, they will not all be orthogonal to each other) it is quite possible that \( S_1 \) (the coefficient associated with the closest vector) will not be larger than both \( S_2 \) and \( S_3 \). Because of the way in which \( S_\beta \) is calculated, as the three library vectors encompassing the desired direction change, there may be a discontinuous jump in the predicted force as \( S_1 \) jumps from the largest coefficient to the second or third largest.

### 4.3.1 2D Implementation

In 2D, interpolating among the library flow directions, \( \theta_\beta \), to match the fluid velocity’s direction, \( \theta_U \), is a much more straight-forward procedure. Since all of our direction vectors lie in a single plane, the interpolation is a simple function of the object’s angle with respect to the inflow, \( \theta_U \), as well as the nearest library angles in either direction, \( \theta_1 \) and \( \theta_2 \) (assume \( \theta_1 \leq \theta_U \leq \theta_2 \), see Figure 4.1):

\[
S_1 = \frac{\cos(\theta_U - \theta_1) - \cos(\theta_2 - \theta_U)\cos(\theta_2 - \theta_1)}{\sin^2(\theta_2 - \theta_1)}, \tag{4.13}
\]

\[
S_2 = \frac{\cos(\theta_2 - \theta_U) - \cos(\theta_U - \theta_1)\cos(\theta_2 - \theta_1)}{\sin^2(\theta_2 - \theta_1)}, \tag{4.14}
\]

where \( S_1 \) and \( S_2 \) are direction dependent weights assigned to each library entry. Note that these equations are simply the solution to the system of equations introduced for planar directions in the 3D case (equations (4.5) and (4.6)).

Interpolating the forces and moments between our library’s fluid speeds will be an identical procedure as was described previously. For a linear interpolation, again use equations (4.7) and (4.8), but for a quadratic interpolation scheme, refer to equations (4.9) and (4.10).
Figure 4.1: Definition of inflow and library angles with respect to an object in 2D (z-dir is positive out of the page)

In either case, the interpolation of our force and moment with respect to direction and speed is simply

\[
F_{avg} = \nu_1(S_1 F_1^1 + S_2 F_2^1) + \nu_2(S_1 F_1^2 + S_2 F_2^2), \quad (4.15)
\]
\[
M_{avg} = \nu_1(S_1 M_1^1 + S_2 M_2^1) + \nu_2(S_1 M_1^2 + S_2 M_2^2). \quad (4.16)
\]

Note that in the 2D case, the moment has only a single component, whereas the force acting in the 2D plane is a vector.

### 4.4 Problem Considerations

B-FOIST is intrinsically well-equipped to handle a broad range of problems, but it is important to know how certain problem characteristics will affect its calculations and results.
4.4.1 Steady vs. Unsteady Flow

Around the Object

B-FOIST is most accurate when solving problems in which flow around the objects is steady. If the object upon which a library is built undergoes severe or erratic shedding, then the temporal averaging of the forces and moments will not accurately reflect the physical situation, leading to possibly poor trajectory predictions. In order for the temporal averaging to be effective, either the object must be massive enough to resist unsteady forces from affecting its overall trajectory, or the oscillation in forces must be relatively small about the temporal average. In other words, the acceleration of the object must be heavily orientation-dependent, and minimally time-dependent in order for B-FOIST to be accurate. Additionally, it is useful for any shedding to be periodic so that bounds for the time-averaging integration can be easily established.

Hence, we expect B-FOIST to be inherently better at simulating three-dimensional problems than two-dimensional problems. An object in a two-dimensional problem is equivalent to a three-dimensional prism of infinite length, which lends itself to pronounced shedding and forces and moments that vary wildly with time. The force history of a typical, low-aspect-ratio, three-dimensional object will tend to be more steady with respect to time.

In the Domain

Steadiness of fluid flow in the domain is also an important consideration. If the flow is relatively steady, then for each time iteration of B-FOIST, the same velocity information from the domain can be used. This will save the trouble and computational cost of loading a new fluid data file for each time iteration.

However, if the flow in the greater domain (especially in the path of the object) depends heavily on time, one should consult a new set of velocity data for each time
step of B-FOIST. This emphasizes the importance of selecting at what point in time the simulation will begin. If one is looking for a specific, temporal flow feature (i.e. moving vortex) to interact with an object, one must carefully consider at what point in the flow history of the greater domain to introduce the object. Of course, this exemplifies another advantage of B-FOIST over traditional FOI solvers, because B-FOIST can be quickly run multiple times to explore a variety of circumstances. A mesh-moving FOI solver, on the other hand, may require several costly simulations before the desired situation is simulated. In fact, a possible application of B-FOIST is to use it hand-in-hand with fully-formulated, mesh-moving FOI solvers. B-FOIST can be inexpensively used to evaluate the behavior of objects for any number of initial conditions, and a more sophisticated FOI solver can be introduced to further simulate only the most interesting cases.

4.4.2 Multiple Objects

While B-FOIST can predict the trajectories of many objects in a flow field, one must remember our assumption that each object is insignificant compared to the flow field and will not influence the fluid flow. Therefore, one must assume that the behavior of each object will not be influenced by nor influence any of the other objects in the problem (we will disregard the possibility of collisions). Now that the objects’ trajectories have been uncoupled due to our assumptions, evaluating a problem with several objects is no different than solving the problem individually for each object under consideration.

For example, if several paratroopers jumped from an airplane in succession (assuming a reasonably long time interval between jumps), it would likely be safe to assume that there would be negligible interaction between them before their parachutes opened. On the other hand, a tube filled with many balls released in a viscous fluid as was simulated by Johnson and Tezduyar [5, 6, 7], would not be a good candidate
for B-FOIST. Because of the proximity of the balls, significant interactions occur between the balls, revealing that each object is indeed coupled by the fluid and cannot be simulated accurately by B-FOIST.

4.4.3 Accounting for an Object's Angular Velocity

Problems in which the object in question travels with a significant angular velocity will not be easily modeled by B-FOIST. This is because our library of forces is compiled by simulating uniform flow over a stationary object, and an object rotating in a fluid is an entirely different phenomenon. There are several ways to approach such problems in order to more accurately simulate an object's behavior. If the object is anticipated to have a moderate angular velocity (“tumbling”), a mesh-moving FOI solver will predict a drag torque as the object tumbles through the fluid. The original B-FOIST formulation will be incapable of predicting this resistance to being rotated in a fluid, although it still may predict the trajectory with reasonable accuracy.

Adding a rotational damping term to B-FOIST may improve our prediction. In addition to calculating forces and moments at different flow directions, one can calculate the moment generated by spinning the object about different axes using a mesh-moving FOI solver. By spinning the object around several axes, in both directions, one can interpolate a damping coefficient, $\eta_{\text{rot}}$. Then the resisting torque is calculated as a function of angular speed and can be incorporated into the normal B-FOIST formulation. For the 2D case, using a symmetric object, it will become

$$M_{\text{damp}} = -\eta_{\text{rot}} \omega$$  \hspace{1cm} (4.17)

where $\eta_{\text{rot}}$ has the dimension of $\frac{[\text{torque}]}{[\text{ang.vel}]}$. In the 3D case, the calculation must be carried out for all three directions in space, with different components of $\eta_{\text{rot}}$ for each axis of rotation. Note that if the object in question is asymmetric, the object must be
rotated in both directions around each axis, since they will incur different rotational drag.

Another problem may involve an object moving with a very high angular velocity ("spinning"). In addition to the drag torques we just discussed, a spinning object will generate extraneous forces that would not be predicted by either the normal B-FOIST formulation, nor by our revised formulation for tumbling objects. To be considered as part of this class of problems, we will assume the object is spinning quickly enough that its angular momentum will preserve the direction and magnitude of the angular velocity throughout the trajectory. Take, for instance, a baseball. Over the course of a baseball’s path from a pitcher’s hand to home plate, it maintains essentially the same angular velocity. In such a problem, the normal library of forces at various directions can be computed while the baseball rapidly spins about its axis. Of course, one again needs to compute library forces and moments with a traditional FOI solver (unless the object is a perfect sphere, and one defines creative boundary conditions), so mesh-moving and remeshing will be required. Utilizing SSMUM may help mitigate these costs.

4.4.4 Uniformity of Fluid Velocity

While B-FOIST is very good at predicting a variety of fluid flow circumstances, it may not accurately model situations in which the motion of the object is influenced by large velocity gradients. In other words, if the flow field is not relatively uniform throughout the anticipated volume of the object, the assumptions regarding the library are invalid and the accuracy of B-FOIST is compromised.

A good example is an object such as a sphere subject to shear flow. Either FOIST or a mesh-moving FOI simulation would show that as the sphere is swept back, it is also rotated because the fluid impinges asymmetrically on the object. B-FOIST, however, would only pick up a single velocity from the larger domain and would have
no way to predict that the sphere would rotate.

Besides shear flow, there are other situations in which flow may not be adequately uniform to use B-FOIST. Vortices of the size on the order of an object’s characteristic length cannot be accounted for by the current B-FOIST implementation.

4.5 Library Considerations

Construction of the library of forces and moments is dictated by the directions and magnitudes of the fluid flow one wishes to examine. While the magnitude selection is a fairly trivial process, an intelligent selection of which directions to include can lead to better results with fewer computations.

4.5.1 Composition of Library

By and large, if one is attempting to limit the number of flow computations needed to make an effective library, resources should be dedicated to increasing the variety of velocity directions rather than diversifying the velocity magnitudes. It is much easier to predict how a change in flow speed affects the resultant forces and moments on an arbitrary object than it is to anticipate how a change in orientation will influence the forces and moments.

Fluid Flow Speeds ($U^\alpha$)

In figure 4.4, we will demonstrate how forces and moments are strongly and predictably related to the fluid flow speed, $U$, over the range of velocities relevant to our
simulation. So strongly in fact, that in most cases (generalized for 2D or 3D)

\[
\frac{(U)^2}{(U^1)^2} \left( \sum_{\beta} S_{\beta} F^{1}_{\beta} \right) \approx \frac{(U)^2}{(U^2)^2} \left( \sum_{\beta} S_{\beta} F^{2}_{\beta} \right) \approx F_{\text{inter}}, \quad (4.18)
\]

\[
\frac{(U)^2}{(U^1)^2} \left( \sum_{\beta} S_{\beta} M^{1}_{\beta} \right) \approx \frac{(U)^2}{(U^2)^2} \left( \sum_{\beta} S_{\beta} M^{2}_{\beta} \right) \approx M_{\text{inter}}. \quad (4.19)
\]

Therefore, substituting into the quadratic form of equation (4.15) or (4.11), we can see

\[
F_{\text{avg}} = \frac{U^2 - U}{U^2 - U^1} F_{\text{inter}} + \frac{U - U^1}{U^2 - U^1} F_{\text{inter}}
\]

\[
= \frac{U^2 - U + U - U^1}{U^2 - U^1} F_{\text{inter}}
\]

\[
= \frac{U^2 - U^1}{U^2 - U^1} F_{\text{inter}} = F_{\text{inter}}. \quad (4.20)
\]

This means that, in effect, it is possible to compose our library of many flow directions, but only a single fluid flow speed, \(U^0\), and still accurately predict the forces and moments acting on an object for a range of flow speeds. If we base our force predictions on this quadratic interpolation, let us generalize our force definition with a vector coefficient:

\[
F^0_{\beta} = C_F^0 (U^0)^2, \quad (4.21)
\]

in which case our formulations become

\[
F_{\text{avg}} = (U)^2 \left( \sum_{\beta} S_{\beta} C_F^0 \right), \quad (4.22)
\]

\[
M_{\text{avg}} = (U)^2 \left( \sum_{\beta} S_{\beta} C_M^0 \right). \quad (4.23)
\]
Fluid Flow Directions ($\epsilon_3$)

As a more fluid directions are added to the library, the ability to predict the moment on an object should generally improve more dramatically than the ability to predict forces. Forces, while they do not necessarily act in the direction of fluid flow, are generally fairly consistent with the fluid velocity. That is to say, one could make a reasonable guess regarding the magnitude and direction of the force on an object knowing the fluid flow and direction, regardless of the object's orientation.

However, without knowing the orientation of the arbitrary object, there is no way to predict the moment that it is being subjected to. This is why having a comprehensive direction library is crucial to determining the moments, and therefore the orientation and trajectory, that an object will experience. Figure 4.5 gives compelling visual evidence that a thin library will indeed most dramatically impact the moment prediction.

4.5.2 Predictable Orientations

For an arbitrary object, it makes sense to evenly distribute the flow directions in the library over the surface of the object. There are certain problems, however, for which the general orientation of an object is known with respect to the flow direction. For instance, given a problem concerning the separation of a missile from an airplane, assuming the problem is constructed correctly, one can be confident that the nose of the missile will always be facing forward. Subsequently, there is no need to calculate directions with the flow facing the tail of the plane, and one can leave these directions out of the force and moment libraries.

Such a situation presents some attractive options depending on whether time-savings or accuracy is of more value. One can conduct fewer fluid flow simulations to form a smaller library devoid of superfluous data for velocities facing the tail. The other option would be to utilize the computational time that would have been used
to calculate unnecessary flow directions to calculate additional library entries in more relevant flow directions.

### 4.5.3 Planes of Symmetry

There are additional ways to limit the number of flow simulations conducted to construct an object’s library. Planes of symmetry are not uncommon in objects that are good candidates for B-FOIST. Any planes of symmetry in an object should be taken advantage of. If a plane of symmetry exists, the calculations done for each library entry only need to be done for one side of the symmetry plane. Simply reversing the sign of the force component outside of the plane will fill out the rest of the library.

Symmetry can also pose problems in certain situations. For instance, if an object has three perpendicular planes of symmetry, one might easily select six perpendicular directions that lie along the intersections of the symmetry planes for the basis of one’s library. However, this would be a critical mistake. If the only directions in a library lie along the intersection of symmetry planes, then no library entry will contain moments (remember, the presence of moments in the library is a direct result of the asymmetry of the object with respect to the direction of flow). Subsequently, no amount of interpolation between those directions would predict any moments about the object. If, during the simulation, the fluid flow was not parallel with one of the six directions, B-FOIST would fail to predict any angular acceleration. Therefore the most obvious choice for one’s directions may not be the wisest. One should ensure that several library directions lie off of symmetric planes so that moments will be accurately predicted in a variety of orientations.

Additionally, suppose the object of interest is axisymmetric and the axis of symmetry is exactly parallel to the direction of fluid flow. A precise library will not predict any torques acting on the object in this situation, and if the flow is steady, the object will not have an opportunity to rotate as its swept back. In reality, some
amount of shedding would likely occur, and a complete finite-element method sim-
ulation would predict moments that would cause the object to rotate. If the object
began in an unstable equilibrium configuration, it would proceed to tumble, but if it
began in stable equilibrium, it would return to its original orientation.

It is suggested that if this is a possibility, a randomness be added to the interpo-
lation of moments, giving the object a chance to rotate slightly off of equilibrium. If the
equilibrium is unstable, the library will now be able to calculate moments that will
continue to rotate the object and allow it to tumble as it would in a mesh-moving FOI
code. If the object is in a stable equilibrium configuration, this small randomness will
not faze the object’s orientation, because the library will predict moments to restore
the equilibrium orientation.
4.6 2D Ellipse Simulation

Our numerical results were obtained for an ellipse immersed in viscous fluid flowing in the positive $x$ direction. The ellipse had a major axis to minor axis ratio of 3:1. In looking to simulate unbounded, uniform flow, we turned to the measurements of Tezduyar et al. [1] to establish the dimensions of our outer domain ensuring it was at least as big as recommended. The parameters of the fluid were as follows:

\begin{align*}
\mu &= 0.01, \\
\rho &= 1.0, \\
U_\infty &= 10.0.
\end{align*} 

\hspace{1cm} (4.24) \hspace{1cm} (4.25) \hspace{1cm} (4.26)

4.6.1 FOI Problem Characterization

In order to validate our B-FOIST results, we must compare them to established, fluid-object interaction simulations. For a fair comparison, the same solver should be used to compile the library for B-FOIST as is used to compute the mesh-moving, FOI simulation.

In our case, the most viable 2D solver did not incorporate the mesh-moving techniques needed to run our FOI comparison. Fortunately, the program was formulated in space–time, so altering the code to accommodate a moving ellipse did not require a complete overhaul.

Instead of using an unstructured mesh and having the added complications of solving for mesh motion or remeshing, we decided to utilize a simpler, proven method, used by Mittal and Tezduyar [8]. For the cost of having to design and build our structured mesh, we will save ourselves the hassles of implementing the changes needed to deal with a deforming, unstructured mesh. Referring to Figure 4.2, we will deform different regions of the structured mesh for displacements and rotations of the ellipse.
Figure 4.2: Diagram of different regions of deformation in structured 2D mesh

For angular displacements, region A will move as if rigidly attached to the ellipse, and none of the elements in the region will be deformed. Not only does this preserve the integrity of the mesh around the surface of the ellipse and maintain consistency among the different simulations, it also provides a transition region between our ellipse and the encompassing circle. In a trend that is consistent with the other regions in our mesh, the elements closer to the ellipse are smaller than those at the edge of the domain. It is important to increase resolution of the mesh near our object to resolve flow characteristics that largely dictate the forces and moments on the ellipse. This is especially important in region A, where the boundary layer should be highly resolved (see Figure 4.3). This circle and the outer, concentric circle enclose region B. Region B consists of a number of concentric rings of elements that will deform in shear as the ellipse rotates. Having many layers of elements in region B allows the mesh to undergo large angular displacements before needing to reconnect the nodes. This is an extension of the original SSMUM. Tezduyar et al. [14] used only a single layer of elements for which the nodes were reconnected often. Region C is another transition region, from the outer circle to the square, that will allow the inner mesh to interface
with the outer regions of our domain. The elements in this region do not deform, but will translate exactly with the ellipse.

Figure 4.3: Close-up view of elements near the edge of angled 2D ellipse

The translation calculations will be straight-forward for our structured mesh, and we will not need to solve any systems of equations. The regions above, below, left and right of the square encompassing the ellipse act like an accordion, growing and shrinking as the square moves around the domain. For instance, as the ellipse moves to the right, sections III, VI, and IX of the mesh will be compressed, while sections I, IV, and VII will be expanded, simply by distributing the displacement amongst each column of elements. Elements will be deformed proportionally according to the size of the element. That is, the smaller elements closer to the ellipse will not be elongated or compressed as must as the bigger elements closer to the edge of the domain. In the meantime, sections II and VIII will of course simply translate, undeformed, to the right. Note that the elements upstream of the ellipse will grow larger and could potentially be stretched very thin, but this will not be very detrimental to our solution because the flow is still uniform in this region, before it reaches the ellipse.
Several adjustments were made while designing our structured mesh. First, we decided it was important to have a higher concentration of nodes at either end of the ellipse, because of the greater rate of change of the radius of curvature of the edge. Also, even when the ellipse was oriented such that it was symmetric about the direction of flow (i.e. no angle of attack), our time-integration of the force resulted in a non-zero lift. It was a small component of the overall force, but enough to raise concerns. Connecting the mesh’s nodes symmetrically about the direction of flow alleviated the problem.

4.6.2 B-FOIST Problem Characterization

The principles espoused in Section 4.5.3 are employed in our 2D ellipse example. We initially selected the fluid flow at an incidence of 45 degrees to garner information about the ellipse’s lift, drag, and moment. In fact, that is the only direction we needed. Because of the two lines of symmetry in our ellipse, our library is able to be filled out in four, perpendicular directions without additional flow computations. And because the flow is not along either line of symmetry, we are able to maximize the amount of information about how our object interacts with the fluid thanks to that one computation. With the one library entry, we calculate a minimum drag and no torque when the ellipse’s major axis is aligned with the direction of flow. We predict a maximum drag, and again no net torque when that same axis is oriented perpendicular to the flow. In this way, we are able to predict the ellipse’s trajectory and orientation to a high degree of accuracy, considering the amount of computational time we invested.

In addition to this most basic library, we calculated forces and moments from flow directions at intervals of 30 and 15 degrees. Again, we only have to numerically calculate these forces in a single quadrant, and because of symmetry, the forces across the remaining 270 degrees can be anticipated.
Figure 4.4 illustrates how accurate quadratic predictions can be when interpolating from only a single library flow speed. An ellipse experienced fluid flow at \( \theta_U = 15^\circ \) and the forces were time-averaged for fluid speeds ranging from \( U = 5.0 \) to \( U = 10.0 \). In the figure, data points of the computed force at different flow speeds are plotted against an interpolation based only on the computed force at \( U = 10.0 \). The quadratic interpolation scheme comes directly from equations (4.22) and (4.23) in Section 4.5.1. It is evident how accurately the interpolated forces mimic the actual, computed forces.

Figure 4.4: Plot of force calculations versus quadratic prediction of force

Thanks to the predictable way in which the force varies with fluid flow speed, all of our library entries for the 2D ellipse case are calculated at a single flow speed, \( U = 10.0 \). The ellipse’s force and moment histories compare favorably to previous work presented by Blodgett in [4]. Refer to Tables 4.1 and 4.2 for the different B-FOIST libraries of the 2D ellipse.

Figure 4.5 show the force and moment predictions for all angles of attack, \( \theta_U \), at \( U = 10.0 \), according to the three different libraries defined in Tables 4.1 and 4.2.
It is evident from the three figures that the moment prediction for the library with 90° intervals is markedly different from the other two libraries. Although the force predictions for each library seem to have the same general shape and behavior, the moment predictions look very different depending on how comprehensive the libraries are.

<table>
<thead>
<tr>
<th>90 degree intervals</th>
<th>30 degree intervals</th>
<th>15 degree intervals</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{F_x}$</td>
<td>$C_{F_y}$</td>
<td>$C_{F_x}$</td>
</tr>
<tr>
<td>$\theta = 0^\circ$</td>
<td>0.2212</td>
<td>0</td>
</tr>
<tr>
<td>$\theta_U = 15^\circ$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\theta_U = 30^\circ$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\theta_U = 45^\circ$</td>
<td>2.5683</td>
<td>-1.7347</td>
</tr>
<tr>
<td>$\theta_U = 60^\circ$</td>
<td>3.6424</td>
<td>-1.8448</td>
</tr>
<tr>
<td>$\theta_U = 75^\circ$</td>
<td>4.3801</td>
<td>0</td>
</tr>
<tr>
<td>$\theta_U = 90^\circ$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: Tabulated force coefficient data for an ellipse at $U = 10.0$

<table>
<thead>
<tr>
<th>90 degree intervals</th>
<th>30 degree intervals</th>
<th>15 degree intervals</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_M$</td>
<td>$C_M$</td>
<td>$C_M$</td>
</tr>
<tr>
<td>$\theta_U = 0^\circ$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\theta_U = 15^\circ$</td>
<td></td>
<td>0.6993</td>
</tr>
<tr>
<td>$\theta_U = 30^\circ$</td>
<td></td>
<td>0.7491</td>
</tr>
<tr>
<td>$\theta_U = 45^\circ$</td>
<td>0.7010</td>
<td>0.7010</td>
</tr>
<tr>
<td>$\theta_U = 60^\circ$</td>
<td>0.7314</td>
<td>0.7314</td>
</tr>
<tr>
<td>$\theta_U = 75^\circ$</td>
<td>0</td>
<td>0.4967</td>
</tr>
<tr>
<td>$\theta_U = 90^\circ$</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.2: Tabulated moment coefficient data for an ellipse at $U = 10.0$

We will also attempt to account for damping in several of our B-FOIST simulations. The damping coefficient was obtained by rotating the ellipse around its centroid at $\omega = 1.0 \frac{\text{[rad]}}{\text{time}}$. Figure 4.6 shows the moment history of the fluid on the ellipse. We can see the solution eventually stabilizes at approximately $M = 2.3 \frac{[\text{torque}]}{[\text{ang.vel}]}$, so our damping coefficient will be $\eta_{\text{rot}} = 2.3 \frac{[\text{torque}]}{[\text{ang.vel}]}$. 
Figure 4.5: Comparison of body force predictions for 2D ellipse using three different libraries
Figure 4.6: Graph of moment history for rotated 2D ellipse used to predict damping coefficient ($\omega = 1.0 \frac{\text{rad}}{\text{time}}$)
4.6.3 Numerical Results

Ellipse Undergoing Translation

In order to properly analyze how B-FOIST predicts the general movement of an object in a flow field, it behooves us to isolate the different aspects of an object’s trajectory. First, we will restrict the ellipse’s ability to rotate, while allowing it to freely translate in our uniform flow field. The ellipse was arbitrarily fixed at an angle of 15 degrees to the inflow. The main goal was to accurately predict vertical and horizontal movement of the ellipse. Note that while the ellipse is fixed at an an angle of attack of 15 degrees, $\theta_U$ will not remain constant. As the ellipse acquires a vertical velocity, the fluid velocity relative to the moving ellipse will change. We will begin our simulation after the fluid flow has been fully developed around the ellipse, and the ellipse will be released from rest.

In order to observe the effect mass has on our ability to predict the ellipse’s trajectory, we will evaluate scenarios of $m = 2500$ and $m = 10000$. Note that with a fluid density of $\rho = 1.0$, an ellipse of mass 2500 and area of 2.4 will be approximately one thousand times denser than the surrounding fluid. In an attempt to, in a sense, normalize the duration of the simulations according to the distance traveled amongst the different ellipses, we must run the heavier ellipse longer than the less massive ellipse. It is not difficult to estimate how the time duration for each new case will
relate a previous simulation:

\[ \Delta x_1 = \Delta x_2 \]
\[ \frac{1}{2} a_1 t_1^2 = \frac{1}{2} a_2 t_2^2 \]
\[ \frac{F}{m_1} t_1^2 = \frac{F}{m_2} t_2^2 \]
\[ \frac{t_2^2}{t_1^2} = \frac{m_2}{m_1} \]
\[ \frac{t_2}{t_1} = \sqrt{\frac{m_2}{m_1}} \]
\[ t_2 = t_1 \sqrt{\frac{m_2}{m_1}} \] (4.27)

Figures 4.7 and 4.8 track the predicted force histories from the various B-FOIST libraries as they compare to the FOI program for ellipses of the two different masses. Figures 4.9 and 4.10 illustrate that our ability to predict the ellipse's translational displacement is independent of its mass for this particular problem. This indicates that the shedding is not severe enough to significantly disturb either of the two ellipses, as they proceed along their rather smooth trajectories.

Each of the figures compares B-FOIST predictions from different libraries, highlighting the necessity of building adequate force libraries.
Figure 4.7: Graphs of force histories for translating 2D ellipses comparing mesh-moving FOI to three different B-FOIST libraries ($m = 2500$)

Figure 4.8: Graphs of force histories for translating 2D ellipses comparing mesh-moving FOI to three different B-FOIST libraries ($m = 10000$)
Figure 4.9: Graphs of displacement histories for translating 2D ellipses comparing mesh-moving FOI to three different B-FOIST libraries \( (m = 2500) \)

Figure 4.10: Graphs of displacement histories for translating 2D ellipses comparing mesh-moving FOI to three different B-FOIST libraries \( (m = 10000) \)
Ellipse Undergoing Rotation

The second behavior isolated was the ellipse's ability to rotate. Pinning the ellipse about its center of mass, we allowed it to rotate freely in the flow field without any ability to translate. In this circumstance, the relative fluid velocity will always be $U_\infty$, but $\theta_U$ will vary as the ellipse tumbles due to the fluid flow. Again, the fluid flow will be fully developed when the ellipse is released from rest at $t = 0$.

We will evaluate the rotating ellipse at inertias of $I = 1000$ and $I = 4000$. To again attempt to normalize the duration of the simulations according to the angular displacement amongst the different ellipses, we will again run the different ellipses for different lengths of time. The math leading up to equation (4.27) will not change in a meaningful way. Therefore, we will normalize the duration of our simulations with the following equation:

$$t_2 = t_1 \sqrt{\frac{I_2}{I_1}}$$  \hspace{1cm} (4.28)

Notice in Figures 4.11 and 4.12 how the moment histories vary among the different simulations. The slower moving, more massive ellipse undergoes a lengthy moment history to rotate the same amount as the less massive ellipse. Therefore, B-FOIST's moment predictions stand a better chance of accurately predicting the motion of a massive object, because through its resistance to acceleration, an object's increased inertia effectively averages the erratic force over time. A visual inspection of Figures 4.13 and 4.14 reveals how the more massive ellipse's angular velocity is smoother, as if responding to a steadier moment than its less massive counterpart. As a direct result, the velocity predictions of the most massive ellipse are more accurate as compared to the FOI results. This ultimately results in the more accurate B-FOSIT trajectory prediction for the more massive ellipse as demonstrated by the results presented in Figures 4.15 and 4.16.

The aforementioned figures present B-FOIST results from variety of libraries plot-
ted against results from our full fluid-object interaction code. Examining the graphs independently of each other illustrates the importance of using detailed body force libraries.

Comparing the behaviors of the different ellipses (as simulated by the FOI program), we can see that the lighter ellipse was damped more quickly than the more massive ellipse. The fact that the lighter ellipse traveled with a higher angular velocity lends credence to our damping coefficient from Section 4.4.3.

Therefore, we proceeded to experiment with incorporating damping into the B-FOIST implementation. The pinned ellipse \((I = 4000)\) was simulated with B-FOIST for both the damped and the undamped case using the most accurate B-FOIST library (force and moment records at 15 degree intervals). Figure 4.17 illustrates how the moment prediction was slightly improved. This translates into very slightly improved predictions for the ellipse trajectory, as illustrated in Figure 4.18.

While the ellipse could not translate in the \(x\) or \(y\) direction, it did experience forces in both directions. Figure 4.19 compares the components of the force on the most massive ellipse with those predicted by the most refined B-FOIST library for the damped case. Their accuracy lends an optimistic outlook for our next task of unfettered motion.
Figure 4.11: Graph of moment histories for rotating 2D ellipses comparing mesh-moving FOI to three different B-FOIST libraries ($I = 1000$)

Figure 4.12: Graph of moment histories for rotating 2D ellipses comparing mesh-moving FOI to three different B-FOIST libraries ($I = 4000$)
Figure 4.13: Graph of angular velocity histories for rotating 2D ellipses comparing mesh-moving FOI to three different B-FOIST libraries \((J = 1000)\)

Figure 4.14: Graph of angular velocity histories for rotating 2D ellipses comparing mesh-moving FOI to three different B-FOIST libraries \((J = 4000)\)
Figure 4.15: Graph of angular displacement histories for rotating 2D ellipses comparing mesh-moving FOI to three different B-FOIST libraries ($I = 1000$)

Figure 4.16: Graph of angular displacement histories for rotating 2D ellipses comparing mesh-moving FOI to three different B-FOIST libraries ($I = 4000$)
Figure 4.17: Graph of moment histories for rotating 2D ellipses comparing mesh-moving FOI to damped/undamped 15° B-FOIST library ($I = 4000$)

Figure 4.18: Graph of angular displacement histories for rotating 2D ellipses comparing mesh-moving FOI to damped/undamped 15° B-FOIST library ($I = 4000$)
Figure 4.19: Graph of force histories for rotating 2D ellipses comparing mesh-moving FOI to damped 15° B-FOIST library ($I = 4000$)
Ellipse Undergoing Rotation and Translation

After isolating the two types of motion experienced by the ellipse, we elected to allow the ellipse to freely move, unrestricted in the flow field. Again, different cases were evaluated, but the mass and inertia were scaled equally to be consistent. In order to mimic our previous simulations, we ran cases for \( m = 2500 \) and \( I = 1000 \) as well as \( m = 10000 \) and \( I = 4000 \). Notice that the time span for each case can be estimated using either equation (4.27) or equation (4.28).

Figures 4.20 and 4.21 illustrate the body forces felt over the course of the three simulations as they compare to the library predictions. Again, the more massive ellipse undergoes the most shedding; its inertia allows it to resist the oscillating forces and move more smoothly and predictably than its less massive counterpart. Figures 4.22 and 4.23 show the ellipse velocities, which more clearly illustrate this point. Notice how oscillations are less erratic for the more massive ellipse.

The accuracy of force prediction allows us to predict the trajectory as the displacement plots in Figures 4.24 and 4.25 indicate. Notice that although the 90° library appears closest to the mesh-moving FOI solver's predictions of the vertical and angular displacements, it is evident that the overall trajectories are worse. Both the angular displacement and velocity graphs show that the other two libraries are predicting the period of the rotation more accurately, which will lead to better predictions as the simulation proceeds. Similarly, the period of the y-velocity prediction using the 90° library is significantly worse than the other two libraries even though it may predict parts of the history more accurately than the other two libraries.

Having repeatedly shown the advantage of predicting the trajectory of a massive object, let us more carefully consider the case of \( m = 10000 \) and \( I = 4000 \). Looking to Figure 4.26, we can see that adding the damping coefficient to our formulation again does relatively little to improve our trajectory predictions.
Figure 4.20: Graph of force histories for free 2D ellipses comparing mesh-moving FOI to three different B-FOIST libraries ($m = 2500$, $I = 1000$)
Figure 4.21: Graph of force histories for free 2D ellipses comparing mesh-moving FOI to three different B-FOIST libraries ($m = 10000$, $I = 4000$)
Figure 4.22: Graph of velocity histories for free 2D ellipses comparing mesh-moving FOI to three different B-FOIST libraries ($m = 2500$, $I = 1000$)
Figure 4.23: Graph of velocity histories for free 2D ellipses comparing mesh-moving FOI to three different B-FOIST libraries \((m = 10000, I = 4000)\)
Figure 4.24: Graph of displacement histories for free 2D ellipses comparing mesh-moving FOI to three different B-FOIST libraries ($m = 2500, I = 1000$)
Figure 4.25: Graph of displacement histories for free 2D ellipses comparing mesh-moving FOI to three different B-FOIST libraries \((m = 10000, I = 4000)\)
Figure 4.26: Graph of displacement histories for free 2D ellipses comparing mesh-moving FOI to undamped/damped 15° B-FOIST library ($m = 10000$, $I = 4000$)
4.7 Future Work

There is much room for improvement of B-FOIST, especially in the way in which it predicts damping effects. It is clear from the fully-coupled FOI simulations that rotational damping is a significant factor in the trajectory of an object, but we were not able to significantly improve our predictions with the implementation of the damping coefficient. If the coefficient were made artificially larger, the predictions will improve. The challenge, however, is to accurately anticipate coefficient values instead of selecting them to match previous results. Then one should consider building a library of damping coefficients for various angular speeds similar to the pre-existing force and moment libraries. It would also be interesting to explore cases where the added-mass effect becomes a significant factor in the dynamics of a fluid–object interactions problem.

Because the coefficient of drag/lift of an object is not independent of the Reynolds number, we know that we cannot quadratically interpolate from library entries at a single flow speed for any possible situation. That is, as the Reynolds number changes dramatically, the flow regime will change, and a simple quadratic interpolation may not suffice. Just as we studied the effect of changing the number of flow directions in our library, more work should be done to determine how changing the number of flow speeds will affect the predictions B-FOIST will make.

Additionally, the ability of B-FOIST to predict 3D motion should be thoroughly explored. We have already stated that we anticipate the proficiency of B-FOIST to improve for 3D trajectories because of the reduction of shedding effects. Research is underway on B-FOIST’s ability to predict the trajectory of a paratrooper falling away from an airplane. Such store separation problems should be exceptionally well-suited for B-FOIST.
Chapter 5

Special Space–Time Formulation

5.1 Introduction

There are many circumstances where fully-coupled fluid–object interactions simulations using mesh-moving should be carried out, despite their burdensome computational costs. Many problems cannot be accurately approximated by B-FOIST; even if a problem can be solved with B-FOIST, it may be beneficial to compare the B-FOIST results with a straight-forward FOI simulation.

When carrying out fully-formulated FOI simulations in space–time, there are opportunities to improve the way shape function derivatives and element-level vectors and matrices are calculated by many codes. The Special Space–Time Formulation (SSTF) eliminates many repetitive or otherwise unnecessary calculations from a solver’s element level matrix/vector formation subroutine.

Because most space–time meshes are structured in time, even as spatial mesh motion occurs, the space–time slab surfaces remain parallel and the slabs retain their uniform thickness. By recognizing that the space–time mesh is thus structured in time, implementing SSTF can result in two types of savings. First, the calculation of shape function derivatives at each Gauss point can be simplified. Second, the
formulations for element-level vectors and matrices can be decomposed and simplified. Because these computations need to be done for each element of the space–time mesh for each iteration of every time step, they can constitute a large portion of the overall CPU time for a problem. Therefore, improvements in the way in which they are calculated can have a significant impact on the program’s overall performance.

SSTF can be seamlessly applied to parallel computations as well, because it is implemented on the element level.

## 5.2 Shape Function Derivatives

SSTF can be formulated by first separating the typical shape function into space and time parts [19]:

\[
N_a^\alpha = N_a T^\alpha = N_a(\xi) T^\alpha(\theta), \quad \alpha = 1, 2, \ldots n_{cn}; \quad \alpha = 1, 2, \quad (5.1)
\]

where

\[
T^1(\theta) = \frac{1}{2}(1 - \theta) \quad (5.2)
\]

\[
T^2(\theta) = \frac{1}{2}(1 + \theta) \quad (5.3)
\]

\[
T_{\theta}^1 = -\frac{1}{2}, \quad (5.4)
\]

\[
T_{\theta}^2 = +\frac{1}{2}. \quad (5.5)
\]

Refer to Figure 5.1 to better visualize how \( \alpha \) and \( \alpha \) relate to the geometry of a typical space–time element.

Note that the spatial shape functions are left undefined because they are rather arbitrary and depend on the number of spatial dimensions as well as the element
Figure 5.1: Drawing of a typical element in space–time type. On the other hand, the temporal shape function can be defined, because the space–time mesh is structured the same regardless of the rest of the mesh.

No matter how the spatial shape functions are defined for an arbitrary element and number of spatial dimensions, they should satisfy the following condition:

\[
\mathbf{x} = \sum_{\alpha=1}^{2} \sum_{a=1}^{n_{en}} N_{\alpha}^{1} \mathbf{x}_{a}^{1} + \sum_{a=1}^{n_{en}} N_{\alpha}^{2} \mathbf{x}_{a}^{2}
\]  

(5.6)

\[
= \sum_{a=1}^{n_{en}} N_{a}^{1} \mathbf{x}_{a}^{1} + \sum_{a=1}^{n_{en}} N_{a}^{2} \mathbf{x}_{a}^{2}
\]  

(5.7)

\[
= \sum_{a=1}^{n_{en}} N_{a} (T^{1} \mathbf{x}_{a}^{1} + T^{2} \mathbf{x}_{a}^{2})
\]  

(5.8)

\[
= \sum_{a=1}^{n_{en}} N_{a} \mathbf{x}_{a}(\theta) = \sum_{a=1}^{n_{en}} N_{a} \mathbf{x}_{a}.
\]  

(5.9)

In addition, we should also establish a definition for \( t \) and how it relates to the
temporal shape functions:

\[
 t = \sum_{\alpha=1}^{2} T^\alpha t^\alpha, \\
 = T^1 t^1 + T^2 t^2. 
\]  

Without making any assumptions about the space–time mesh’s structure in time, one would normally have to calculate the inverse of an \((n_{sd} + 1) \times (n_{sd} + 1)\) matrix in order to obtain the derivatives of the shape function with respect to the global coordinates. Implementing SSTF would effectively reduce the problem to taking the inverse of an \(n_{sd} \times n_{sd}\) matrix. Because the inverse calculation is not a trivial computation, reducing the matrix to \(n_{sd} \times n_{sd}\) may result in significant computational savings. We will examine the calculations involved for \(n_{sd} = 2\).

\[
\begin{pmatrix}
 N^\alpha_{a,x} \\
 N^\alpha_{a,y} \\
 N^\alpha_{a,t}
\end{pmatrix} =
\begin{pmatrix}
 \xi_x & \eta_x & \theta_x \\
 \xi_y & \eta_y & \theta_y \\
 \xi_t & \eta_t & \theta_t
\end{pmatrix}^{-1}
\begin{pmatrix}
 N^\alpha_{a,\xi} \\
 N^\alpha_{a,\eta} \\
 N^\alpha_{a,\theta}
\end{pmatrix} = \mathbf{R}^{ST}
\begin{pmatrix}
 N^\alpha_{a,\xi} \\
 N^\alpha_{a,\eta} \\
 N^\alpha_{a,\theta}
\end{pmatrix},
\]  

\[
\begin{pmatrix}
 N^\alpha_{a,x} \\
 N^\alpha_{a,y} \\
 N^\alpha_{a,t}
\end{pmatrix} =
\begin{pmatrix}
 x_\xi & y_\xi & t_\xi \\
 x_\eta & y_\eta & t_\eta \\
 x_\theta & y_\theta & t_\theta
\end{pmatrix}^{-1}
\begin{pmatrix}
 N^\alpha_{a,\xi} \\
 N^\alpha_{a,\eta} \\
 N^\alpha_{a,\theta}
\end{pmatrix} = (\mathbf{Q}^{ST})^{-1}
\begin{pmatrix}
 N^\alpha_{a,\xi} \\
 N^\alpha_{a,\eta} \\
 N^\alpha_{a,\theta}
\end{pmatrix}. 
\]  

The critical assumption that the mesh is structured in time results in \(t_\xi = t_\eta = 0\) because, of course, \(t\) does not vary with respect to space. We also know that \(t_\theta = \frac{\Delta t}{2}\).

This simplifies the \(\mathbf{Q}^{ST}\) matrix as follows:

\[
\mathbf{Q}^{ST} =
\begin{pmatrix}
 x_\xi & y_\xi & 0 \\
 x_\eta & y_\eta & 0 \\
 x_\theta & y_\theta & \frac{\Delta t}{2}
\end{pmatrix}. 
\]  

Now, taking the inverse becomes nearly equivalent to taking the inverse of a 2 × 2
matrix, as opposed to the more complicated \(3 \times 3\) matrix inverse. Similarly, a \(4 \times 4\) matrix can be simplified to a \(3 \times 3\) matrix. To take the inverse of a \(2 \times 2\) matrix, we must solve the determinant of an \(2 \times 2\) matrix. The determinant and inverse are found to be, respectively:

\[
det Q^{ST} = \frac{\Delta t}{2} \det \begin{pmatrix} x, \xi & y, \xi \\ x, \eta & y, \eta \end{pmatrix} = \frac{\Delta t}{2} J(\xi, \eta, \theta) = \frac{\Delta t}{2} J, \quad (5.15)
\]

\[
(Q^{ST})^{-1} = R^{ST} = \frac{1}{\frac{\Delta t}{2} J} \begin{pmatrix} \frac{\Delta t}{2} y, \eta & -\frac{\Delta t}{2} y, \xi & 0 \\ -\frac{\Delta t}{2} x, \eta & \frac{\Delta t}{2} x, \xi & 0 \\ -x, \theta y, \eta + y, \theta x, \eta & x, \theta y, \xi - y, \theta x, \xi & J \end{pmatrix}. \quad (5.16)
\]

We can rewrite \(R^{ST}\) as

\[
R^{ST} = \begin{pmatrix} \frac{1}{J} \begin{pmatrix} y, \eta & -y, \xi \\ -x, \eta & x, \xi \end{pmatrix} & 0 \\ \frac{-x, \theta y, \eta + y, \theta x, \eta}{\frac{\Delta t}{2} J} & \frac{x, \theta y, \xi - y, \theta x, \xi}{\frac{\Delta t}{2} J} \end{pmatrix}, \quad (5.17)
\]

and define

\[
R = \frac{1}{J} \begin{pmatrix} y, \eta & -y, \xi \\ -x, \eta & x, \xi \end{pmatrix}, \quad (5.18)
\]

\[
V_\xi = \frac{-x, \theta y, \eta + y, \theta x, \eta}{\frac{\Delta t}{2} J}, \quad (5.19)
\]

\[
V_\eta = \frac{x, \theta y, \xi - y, \theta x, \xi}{\frac{\Delta t}{2} J}. \quad (5.20)
\]

Already, by examining the simplicity of \(R^{ST}\), one can see the computational savings afforded by rewriting the shape function derivative calculations. To reiterate, the improvement in efficiency would be even more dramatic for the 3D case.

Let us define the derivatives of the global coordinates with respect to the local
coordinates in terms of the shape functions:

\[ x_\xi = \sum_{a=1}^{n_{\text{en}}} N_{a,\xi} x_a(\theta), \quad (5.21) \]

\[ x_\eta = \sum_{a=1}^{n_{\text{en}}} N_{a,\eta} x_a(\theta), \quad (5.22) \]

\[ x_\theta = \sum_{a=1}^{n_{\text{en}}} N_a \frac{x_a^2 - x_a^1}{2} = \sum_{a=1}^{n_{\text{en}}} N_a \frac{\Delta x_a}{2}. \quad (5.23) \]

Rewriting the derivative with respect to \( \theta \) after dividing by \( \frac{\Delta t}{2} \) yields

\[ \frac{x_\theta}{\frac{\Delta t}{2}} = \sum_{a=1}^{n_{\text{en}}} N_a \frac{\Delta x_a}{\Delta t}. \quad (5.24) \]

We will define the nodal mesh velocity, \( V_a = \frac{\Delta x_a}{\Delta t} \). Therefore, the interpolated mesh velocity will simply be \( \mathbf{V} = \frac{x_\theta}{\frac{\Delta t}{2}} = \sum_{a=1}^{n_{\text{en}}} N_a \frac{\Delta x_a}{\Delta t} \). We can now define \( V_\xi \) and \( V_\eta \) in terms of the mesh velocity for use with defining the time derivative of the shape function:

\[ \begin{pmatrix} V_\xi \\ V_\eta \end{pmatrix} = -\mathbf{V} \cdot \mathbf{R}. \quad (5.25) \]

Now, referring back to equation (5.12), we can rewrite our shape function derivative definitions as

\[ \nabla N^\alpha_a = \mathbf{R} \cdot \nabla_\xi N^\alpha_a, \quad (5.26) \]

\[ N_{a,t}^\alpha = (-\mathbf{V} \cdot \mathbf{R}) \cdot \nabla_\xi N^\alpha_a + \frac{2}{\Delta t} N_{a,\theta}^\alpha. \quad (5.27) \]

Using equation (5.1), we will separate the shape functions into space and time parts as follows:

\[ \nabla N^\alpha_a = \mathbf{R} \cdot (\nabla_\xi N_a) T^\alpha, \quad (5.28) \]

\[ N_{a,t}^\alpha = (-\mathbf{V} \cdot \mathbf{R}) \cdot (\nabla_\xi N_a) T^\alpha + \frac{2}{\Delta t} N_a T_{\theta}^\alpha. \quad (5.29) \]
By substituting $\alpha = 1, 2$ and using equations (5.4) and (5.5) to resolve $T_{\phi}^a$, we can more explicitly write the derivatives:

\[
\begin{pmatrix}
\nabla N^1_a
\n\\nabla N^2_a
\end{pmatrix} = R \cdot (\nabla_{\xi} N_a) \begin{pmatrix}
T^1 \\
T^2
\end{pmatrix}, \tag{5.30}
\]

\[
\begin{pmatrix}
N^1_{a,t} \\
N^2_{a,t}
\end{pmatrix} = (-V \cdot R) \cdot (\nabla_{\xi} N_a) \begin{pmatrix}
T^1 \\
T^2
\end{pmatrix} + \frac{N_a}{\Delta t} \begin{pmatrix}
-1 \\
1
\end{pmatrix}. \tag{5.31}
\]

Again, it is evident that there are far fewer operations to perform than would be required for the inverse of the full $3 \times 3$ matrix.

### 5.3 Element Vector Definition

Besides savings from reformulating the shape function derivatives, there are also savings to be had by reformulating the way in which the element-level matrices and/or vectors are calculated. When formulating element-level matrices or vectors, one must integrate each term over all of the Gauss points in space and time for each shape function. For an element-level vector, we will examine a fairly generic integral term $N^\alpha_a y$, for which the integration would be

\[
\int_Q N^\alpha_a y dQ = \sum_{j=1}^{n_{int}^t} \sum_{k=1}^{n_{int}^s} \left( N^\alpha_a y \frac{\Delta t}{2} J \right)_{\xi_k, \phi_j} W^{ST}_{kj}, \quad a = 1...n_{en}, \quad \alpha = 1, 2, \tag{5.32}
\]

where $n_{int}^t$ is the number of Gaussian integration points in time, $n_{int}^s$ is the number of Gaussian integration points in space, $y$ is any combination of fluid constants and variables, and $W^{ST}_{kj}$ are the integration weights in the parent space–time domain. We will assume $n_{int}^t = 2$ and use $n_{int}$ in place of $n_{int}^t$.

By equation (5.1), we can separate the time and space portions of the shape function (Note, if the gradient of the shape function were being integrated, we would
simply use equation (5.28) to separate the shape function instead:

$$\int_Q N^a_{\alpha} y dQ = \sum_{j=1}^{2} \sum_{k=1}^{\text{n}_{\text{int}}} \left( N_a(\xi_k) T^\alpha(\theta_j) y \frac{\Delta t}{2} J \right)_{\xi_k, \theta_j} W_k. \tag{5.33}$$

Here $W_k$ are the integration weights in the parent spatial domain. The two integration weights in the parent time domain are both unity. Because $T^\alpha(\theta_j)$ is independent of $\xi_k$, we can take it out of the inner $k$ summation:

$$\int_Q N^a_{\alpha} y dQ = \sum_{j=1}^{2} \sum_{k=1}^{\text{n}_{\text{int}}} (N_a y J)_{\xi_k, \theta_j} W_k \frac{\Delta t}{2} T^\alpha(\theta_j), \quad a = 1 \ldots \text{n}_e, \quad \alpha = 1, 2. \tag{5.34}$$

Expanding helps elucidate our potential for savings:

$$\int_Q N^a_{\alpha} y dQ = \left[ \sum_{k=1}^{\text{n}_{\text{int}}} (N_a y J)_{\xi_k, \theta_1} W_k \right] \frac{\Delta t}{2} T^\alpha(\theta_1) + \left[ \sum_{k=1}^{\text{n}_{\text{int}}} (N_a y J)_{\xi_k, \theta_2} W_k \right] \frac{\Delta t}{2} T^\alpha(\theta_2), \quad a = 1 \ldots \text{n}_e. \tag{5.35}$$

It is clearer that now that with SSTF, an integral (summation of $2 \times \text{n}_{\text{int}}$) need only to be computed $\text{n}_e$ times rather than $\text{n}_e \times 2$ times as was done in equation (5.32).

It is worth noting that the SSTF formulation of an element-level matrix will involve $\text{n}_e \times \text{n}_e$ integral computations, whereas the original formulation will require $(\text{n}_e \times 2) \times (\text{n}_e \times 2)$, or four times as many integral computations (see Appendix A for a detailed example).

### 5.3.1 Derivatives with Respect to Time

The previous example was fairly straight-forward, and it adequately represents a majority of the vectors and matrices that need to be calculated when forming global equations for a finite element problem. However, the SSTF formulation for an integral involving the time derivative of a shape function is fundamentally different and slightly
more complicated, so we will give it further attention. Therefore, as another example, we consider the integration of a term involving the time-derivative of a shape function:

$$
\int_Q N_{a,t}^\alpha y dQ = \sum_{j=1}^2 \sum_{k=1}^{n_{\text{int}}} \left( N_{a,t}^\alpha \frac{\Delta t}{2} J \right) \left| \frac{x_k}{\xi_k, \tilde{\theta}_j} \right| W_k, \quad a = 1 \ldots n_{\text{en}}, \quad \alpha = 1, 2.
$$  \hfill (5.36)

Now separating the shape and time functions will not be as simple as $N_a^\alpha = N_a T^\alpha$. Instead, we must refer back to equation (5.29). Substituting, our integral becomes

$$
\int_Q N_{a,t}^\alpha y dQ = \sum_{j=1}^2 \sum_{k=1}^{n_{\text{int}}} \left[ \left( -V \cdot R \right) \cdot \left( \nabla_x N_a \right) T^\alpha + \frac{2}{\Delta t} N_a T^\alpha_\theta \right] \frac{\Delta t}{2} J \left| \frac{x_k}{\xi_k, \tilde{\theta}_j} \right| W_k
$$  \hfill (5.37)

$$
= \sum_{j=1}^2 \sum_{k=1}^{n_{\text{int}}} \left[ \left( -V \cdot R \right) \cdot \left( \nabla_x N_a \right) y J \right] \frac{\Delta t}{2} T^\alpha_\theta \left( \tilde{\theta}_j \right)
$$

$$
+ \sum_{j=1}^2 \sum_{k=1}^{n_{\text{int}}} \left( N_a y J \right) \frac{\Delta t}{2} T^\alpha_\theta \left( \tilde{\theta}_j \right).
$$  \hfill (5.38)

We already know the values of $T^\alpha_\theta \left( \tilde{\theta}_j \right)$ from equations (5.4) and (5.5), so once we substitute $j = 1, 2$ we're left with an expanded equation comparable to equation (5.35):

$$
\int_Q N_{a,t}^\alpha y dQ = \sum_{k=1}^{n_{\text{int}}} \left( -V \cdot R \right) \cdot \left( \nabla_x N_a \right) y J \left| \frac{x_k}{\xi_k, \tilde{\theta}_1} \right| W_k \frac{\Delta t}{2} T^\alpha_\theta \left( \tilde{\theta}_1 \right)
$$

$$
+ \sum_{k=1}^{n_{\text{int}}} \left( N_a y J \right) \left| \frac{x_k}{\xi_k, \tilde{\theta}_1} \right| W_k \frac{-1}{2} \alpha
$$

$$
+ \sum_{k=1}^{n_{\text{int}}} \left( -V \cdot R \right) \cdot \left( \nabla_x N_a \right) y J \left| \frac{x_k}{\xi_k, \tilde{\theta}_2} \right| W_k \frac{\Delta t}{2} T^\alpha_\theta \left( \tilde{\theta}_2 \right)
$$

$$
+ \sum_{k=1}^{n_{\text{int}}} \left( N_a y J \right) \left| \frac{x_k}{\xi_k, \tilde{\theta}_2} \right| W_k \frac{-1}{2} \alpha.
$$  \hfill (5.39)
5.4 SSTF Implementation

Now that we have thoroughly formulated SSTF, we can turn our attention towards implementation. While implementations will vary from code to code, we will attempt to broadly address the problem in order to make it clear how to apply the formulation to any computer program.

Whether one is computing element-level matrices or vectors, the subroutine to define the element-level matrix/vector will likely have the following structure:

Element-Level Matrix/Vector Pseudocode

A section computations:
- initialize arrays
- check boundary conditions
- map nodes
- calculate tau

doj = 1, # of gauss points in time

dok = 1, # of gauss points in space

B section computations:
- calculate derivatives of shape functions
- compute velocities and pressures
- compute derivatives of velocities and pressures
doi = 1, # of shape functions in space--time

(do hi = 1, # of shape functions in space--time)

C section computations:
- sum contributions to form element-level vectors (or matrices)

(end do)

end do

end do
end do

D section computations:
  - any necessary post-processing
    (i.e. assembling element-level matrices/vectors)

The above algorithm can describe either the construction of element-level matrices or vectors. Note that the “do loop” in parentheses is an appropriate addition specific to element-level matrix calculations.

Note that this code segment is carried out for every element in the mesh and every time step (and in vector-based calculations, every iteration). In a typical finite element program, this portion of code constitutes a significant percentage of the overall computations done to solve the system. In fact, it can account for more than three quarters of the CPU time for a finite element code. The motivation for improving the efficiency in any section of the pseudocode should now be clear. SSTF aims at trimming unnecessary fat from sections B and C, which will likely consume most of the computational resources in this code segment.

5.4.1 Implementing Shape Functions Derivatives

In Section 5.2 we described how SSTF improves the efficiency of calculating the derivatives of the shape functions. These improvements will cut into the number of operations carried out in section B. Our implementation will not depend on whether or not we are dealing with element-level vectors or matrices. Benefits can be reaped in several operations-saving applications of SSTF.

First, one should not bother with calculating $t_\xi$ or $t_\theta$ which we know to be 0 and $\frac{\Delta r}{2}$ respectively. This will reduce by $\frac{1}{n_{sd} + 1}$ the number of operations dedicated to calculating derivatives of nodal coordinates with respect to local coordinates.

Second, the derivatives of the shape functions with respect to the global coordinate system and time should not be computed by taking the inverse of an $(n_{sd}+1) \times (n_{sd}+1)$
matrix. Instead, equations (5.30) and (5.31) can be used in lieu of taking the inverse of the $Q^{ST}$ matrix. Indeed, looking at equation (5.17), if we were blindly taking the inverse of $Q^{ST}$, all of the entries in $R^{ST}$ would be full, and involve the same operations as take place in only $[3,1]$ and $[3,2]$ of the new $R^{ST}$. This translates into three times as many operations as the SSTF formulation.

Additionally, regardless of the matrix size, SSTF will always simplify the inverse matrix calculations for all entries outside of the last row, in addition to $t_{g}$. So increasing the matrix will actually result in larger savings. For example, if we worked with $n_{sd} = 3$, our matrix would be $4 \times 4$, and taking the full inverse would involve four times as many operations as the SSTF formulation.

Finally, one will save operations to a lesser extent when the gradients of the shape functions are computed. Normally, multiplying the inverse matrix with the vector of derivatives of shape functions with respect to local coordinates involves a dot product of two vectors of length $n_{sd} + 1$ for each entry. However, because $t_{e} = 0$, one will only be computing the dot product of vectors of length $n_{sd}$.

Depending on the formulation of the fluid solver, there may be a large number of velocity and pressure calculations in section B as well. They will be used in the calculation of the element-level vectors or matrices. If the velocity and pressure computations make up a significant portion of the operations, the SSTF improvement to the derivatives of the shape functions will not result in a significant decrease in terms of percentage of computational time for section B.

### 5.4.2 Implementing Element Vector Definition

Only very minor modification of code is needed to implement SSTF and improve the efficiency of part B in our example algorithm. In contrast, implementing the portion of the SSTF formulation which revises the element-level matrix or vector calculation can be complicated and must be preceded by a thorough understanding
of the algorithm.

We will turn back to our previous example of a generic integral for an element-level vector from equation (5.32). The formulation for such a generic element-level vector will look similar to this:

$$\int_Q N_a^\alpha y dQ = \left[ \sum_{j=1}^{2} \sum_{k=1}^{n_{int}} \left( N_a^1 y J^\frac{\Delta t}{2} |_{\tilde{\xi}_k, \tilde{\eta}_j} W_k \right) \right]_q, \quad a = 1...n_{en}. \quad (5.40)$$

Note, the vector length is $2 \times n_{en}$. Refer to our example algorithm. Here, our double summation over $2 \times n_{int}$ represents the looping of $j$ and $k$ over the Gaussian integration points. The $i$ loop scrolls through the vector, covering all of the shape functions for $a = 1...n_{en}, \alpha = 1, 2$.

In cutting down the size of section C in the algorithm, we will cut the $i$ loop in half (and for matrix calculations, the $h$ loop is cut in half as well). We will not sum any terms depending on $\alpha$ but simply loop over $i = 1...n_{en}$. This is the difference between looping over the number of nodes in a space–time element versus the number of nodes in a space element. $Y_j$ is a vector of length $n_{en}$ where the value of $a$ corresponds to the vector index. Notice that our new vector is of the same length as the vector in equation (5.40). However, instead of summing $j = 1, 2$, we will now store $j = 1$ and $j = 2$ in different parts of the vector:

$$Y = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \left[ \sum_{k=1}^{n_{int}} \left( N_a y J |_{\tilde{\xi}_k, \tilde{\eta}_1} W_k \right) \right]_q. \quad (5.41)$$

To complete our implementation, in section D we must recompile our $Y$ vectors and finally come back to calculate our integral to get a solution identical to equation (5.40):

$$\int_Q N_a^\alpha y dQ = \begin{bmatrix} Y_1 \Delta t^2 T^1(\hat{\theta}_1) + Y_2 \Delta t^2 T^1(\hat{\theta}_2) \\ Y_1 \Delta t^2 T^2(\hat{\theta}_1) + Y_2 \Delta t^2 T^2(\hat{\theta}_2) \end{bmatrix}. \quad (5.42)$$
The implementation changes very little if $Y_j$ contains derivatives of shape functions with respect to space. For more details in implementing a matrix-based code, refer to Appendix A.

For a typical integral, this process cuts the number of operations in section C in half, while adding a small number to section D of our algorithm. Keep in mind that C is still performed once for every integration point, while D is done once to wrap everything up at the end. If there are many vectors being evaluated in C, then implementing SSTF will surely make a significant difference in computational time.

**Derivatives with Respect to Time**

Recall from equation (5.31) that taking the time derivative of a shape function results in an expression from which $T^\alpha$ cannot be factored out. Instead, we will have to pull the time shape functions out of each term in equation (5.31). Now each term must be calculated and stored separately. Still we only loop $i = 1...n_{en}$, but we must now perform additional calculations within the loop. Inside section C, our new intermediate vector will resemble the $Y$ from equation (5.41):

$$
Y = \begin{bmatrix}
Y^i_1 \\
Y^i_1 \\
Y^i_2 \\
Y^i_2
\end{bmatrix}
= \begin{bmatrix}
\sum_{k=1}^{n_{int}} ((-V \cdot R) \cdot (\nabla_{\xi} N_{\alpha}) y J) |_{\xi_k, \tilde{\theta}_1} W_k \\
\sum_{k=1}^{n_{int}} (N_{\alpha} y J) |_{\xi_k, \tilde{\theta}_1} W_k \\
\sum_{k=1}^{n_{int}} ((-V \cdot R) \cdot (\nabla_{\xi} N_{\alpha}) y J) |_{\xi_k, \tilde{\theta}_2} W_k \\
\sum_{k=1}^{n_{int}} (N_{\alpha} y J) |_{\xi_k, \tilde{\theta}_2} W_k
\end{bmatrix}. \quad (5.43)
$$

Note that this vector is twice the length of the original. Instead of being length $2n_{en}$, we are now dealing with a vector of length $4n_{en}$. This will minimally increase the number of operations in section A of our algorithm by initializing a larger vector. Additionally, we must now calculate $-V \cdot R$ in section B of our pseudocode, further cutting into the computational savings from SSTF.
Again, post-processing is needed, and the final integral must be assembled in $D$:

$$
\int_Q N^\alpha a_y dQ = \left[ Y^{i_1 \Delta t} T^1(\tilde{\theta}_1) + Y^{i_1 \Delta t} T^1(\tilde{\theta}_2) + Y^{i_2 \Delta t} T^1(\tilde{\theta}_2) + Y^{i_2 \Delta t} T^1(\tilde{\theta}_2) \right].
$$

(5.44)

This equation resembles equation (5.42), but there are now more operations. Obviously we cannot hope to do any better than to break even with this part of the SSTF implementation. Our overall element-level matrix/vector formation savings will be a result of reformulating the integral calculations which contain no derivatives with respect to time.

\section{5.5 SSTF Results}

While we thoroughly understand how SSTF is designed to speed up finite element computations, it is difficult to analytically predict the expected overall computational savings.

In fact, we expect the computational savings to vary wildly from program to program. The improvement in efficiency will be a function of the structure of the solver, namely, how the element-level matrices and vectors are computed.

We implemented SSTF with the same serial, 2D fluid–object interactions code previously used to calculate the ellipse problem in Section 4.6.3. Indeed, the test problem we will use to evaluate the time savings of SSTF was an unrestrained ellipse undergoing motion in a uniform flow field.

It is important to note that in order to observe the duration of different parts of our program, we are influencing the timing because we are required to add additional lines of code. We anticipate that these disturbances are minor and evenly distributed among the different segments of the program we are interested in.

First, we will look at the breakdown of required CPU time amongst the different
portions of our formation subroutine: A, B, C and D (refer to the pseudocode from Section 5.4). Figures 5.2 and 5.3 show the decomposition of the execution time among the sections of code for the normal and SSTF versions of the vector-based program, respectively.

Notice that most of the time savings enjoyed by the SSTF version of the code are a result of the reduction in the duration of section C of the subroutine. It is not cut by the ideal 50% for the vector case because of the extra time-derivative computations required inside the loop. Also, section D now takes slightly longer because of the additional post-processing required. Section B actually holds quite steady, as apparently our gains from distilling the shape function derivative calculations were offset by the additional computation of $-\mathbf{V} \cdot \mathbf{R}$ needed for time derivatives. If this were a 3D code, the gains from calculating the $4 \times 4$ matrix inverse would likely more than offset this additional computation.

In order to look at the execution time of the overall code, we removed many of the timing commands put in place to analyze the subroutine. Removing these extraneous commands helps us see a more accurate picture of the computational time of a typical run. We timed the entire code, in addition to the program’s initialization subroutines and the element-level vector formation subroutine from Section 5.4. Refer to Table 5.1 for timing data on short-duration simulations of the vector-based version of the code. Figures 5.4 and 5.5 graphically show the CPU time needed as listed by Table 5.1 for a single time step. Similarly, Figures 5.6 and 5.7 represent $nts = 5$ in Table 5.1. The effect of SSTF on the duration of different portions of the flow solver is detailed in Table 5.2.
Figure 5.2: Relative duration of different sections of the subroutine calculating element-level vectors

Figure 5.3: Relative duration of different sections of the subroutine calculating element-level vectors (SSTF)
| nts | **Regular** | | **SSTF** | | | Vector Time | Vector Time | Vector Time | Vector Time |
|-----|-------------|-------|------------|------------|------------|
|     | Formation   | Total  | Formation  | Total  | Formation  | Total  |
| 1   | 57.88       | 95.71  | 43.24      | 81.01    |            |       |
| 2   | 115.09      | 184.16 | 85.87      | 154.82   |            |       |
| 3   | 171.29      | 271.00 | 128.93     | 228.00   |            |       |
| 4   | 228.24      | 360.00 | 171.85     | 302.00   |            |       |
| 5   | 288.96      | 452.04 | 214.68     | 376.18   |            |       |

Table 5.1: Timings of vector formation program with and without SSTF implementation

<table>
<thead>
<tr>
<th></th>
<th>SSTF duration as a percentage of original vector-based program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section A</td>
<td>100%</td>
</tr>
<tr>
<td>Section B</td>
<td>100%</td>
</tr>
<tr>
<td>Section C</td>
<td>60%</td>
</tr>
<tr>
<td>Section D</td>
<td>200%</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>Subroutine</td>
<td>77%</td>
</tr>
<tr>
<td>(A+B+C+D)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Entire</td>
<td></td>
</tr>
<tr>
<td>Flow Solver</td>
<td>84%</td>
</tr>
<tr>
<td>(nts &gt;&gt; 1)</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2: Durations of various components of SSTF 2D vector-based flow solver with respect to the original program
Figure 5.4: Execution times for components of vector-based 2D flow solver ($nts = 1$)

Figure 5.5: Execution times for components of vector-based 2D flow solver with SSTF implementation ($nts = 1$)
Figure 5.6: Execution times for components of vector-based 2D flow solver ($nt_s = 5$)

Figure 5.7: Execution times for components of vector-based 2D flow solver with SSTF implementation ($nt_s = 5$)
5.6 Future Work

While we have fairly extensively evaluated the implementation of SSTF on our 2D vector-based program, there is still much work to do. Applying SSTF to a 3D solver (4D in space–time) will likely be a fruitful endeavor. Also, we still do not know much about how it would affect the performance of a matrix-based solver. It is important to remember that the implementation is going to vary from program to program depending on the way in which the element-level vectors or matrices are formed, therefore, work is not necessarily done with 2D vector-based codes, either.

Additionally, equation (5.34) sheds light on an intriguing possibility. The double integration unambiguously shows that the integration consists of two spatial integrations performed at different time levels ($\theta_1$ and $\theta_2$) within our space–time slab. Further work may involve replacing the inner spatial summation over $n_{int}$ Gauss points with an exact analytical expression for the spatial element (a triangle or tetrahedron). We cannot reduce the entire space–time integral to an analytical expression, because numerical integration is a necessity when distorted wedges arise.

Referring again to the pseudocode, the outer $k$ loop over $n_{int}$ will be omitted, and sections B and C of the program will subsequently be executed $\frac{1}{n_{int}}$ as frequently. Inside the assembly loop, however, we must essentially revert back to a matrix-based code, as the shape functions from the fluid variables will no longer be benign coefficients hidden in $y$. More research is needed to explore which parts of these new matrices can or should be stored, in an effort to balance computational speed with memory requirements.
Chapter 6

Conclusions

We have outlined two different methods for new and innovative approaches to fluid-object interactions problems using space-time computations.

6.1 Beyond FOIST

B-FOIST is a simplification of FOIST, with several advantages over both its predecessor and fully-coupled, mesh-moving formulations.

Through comparison with a mesh-moving, 2D fluid dynamics solver, we were able to see the adeptness with which B-FOIST predicts the forces on a moving object. It can also predict the moment applied by the fluid, although to a lesser degree of accuracy. This is likely due to the fact that the fluid forces rely heavily on the speed and direction of the flow, while the moments are a function of an object’s geometry relative to the fluid flow, which is harder to predict.

Over the course of short simulations, B-FOIST was able to fairly accurately predict the trajectory of a moving ellipse in a uniform flow field, whether the motion is pure translation, pure rotation, or a combination of both. Even when the magnitude of rotation was not predicted extremely accurately, B-FOIST was able to predict the period of the rotation.
Among its drawbacks, we know from the formulation that B-FOIST will not predict the trajectory of a problem dominated by large fluid velocity gradients (e.g. shear flow) with any accuracy. It is also evident from our 2D results that the trajectory of more massive objects is more predictable than their less massive counterparts. More massive objects tend to resist such short-term, transient effects as shedding. Any long-term effects shedding has on the trajectory of an object will be entirely neglected by the B-FOIST prediction, due to its static library.

6.2 Special Space–Time Formulation

As we have discussed before, SSTF improves finite-element solvers in two different ways.

While the shape function derivative calculations were simplified in our 2D program, we did not see appreciable time savings. This may be because additional time derivative calculations offset those savings. We anticipate that in a 3D solver, the shape function derivative modifications will provide additional time savings.

Restructuring the way in which element-level vectors and matrices are calculated has a greater potential to cut computational time, but it much more invasive to the program. In order for it to be a worthwhile procedure, the program must execute a significant number of operations within the space–time shape function loop, because that is the loop which will be cut in half.

One should attempt to anticipate the possible computational savings, and then weigh the desired computational savings with the difficulty of implementation before attempting to modify a fluid solver.
Appendix A

SSTF Element-Level Matrix

A.1 Element Matrix Definition

For an element-level matrix, we’ll examine a fairly generic integral term \( N_a^\alpha N_b^\beta y \), for which the integration would be

\[
\int_Q N_a^\alpha N_b^\beta y dQ = \sum_{j=1}^{n_{int}} \sum_{k=1}^{n_{int}} \left( N_a^\alpha N_b^\beta y \frac{\Delta t}{2} J \right) \bigg|_{\tilde{\xi}_k, \tilde{\eta}_j} W_{k,j}^{ST}, \quad \alpha, \beta = 1, 2. \tag{A.1}
\]

As was done for the element-level vector case, we will assume \( n_{int}^t = 2 \) and use \( n_{int} \) in place of \( n_{int}^{sp} \). Again, \( W_j \) are both unity, therefore we will replace \( W_{k,j}^{ST} \) with \( W_k \).

By equation (5.1) we can separate the time and space portions of the shape function:

\[
\int_Q N_a^\alpha N_b^\beta y dQ = 2 \sum_{j=1}^{n_{int}} \sum_{k=1}^{n_{int}} \left( N_a(\xi_k) T^\alpha(\theta_j) N_b(\xi_k) T^\beta(\theta_j) y \frac{\Delta t}{2} J \right) \bigg|_{\tilde{\xi}_k, \tilde{\eta}_j} W_k. \tag{A.2}
\]

Because \( T^\alpha(\tilde{\theta}_j) T^\beta(\tilde{\theta}_j) \) is independent of \( \tilde{\xi}_k \), we can take it out of the inner summation.
over \( k \):

\[
\int_Q N_a^\alpha N_b^\beta y dQ = \sum_{j=1}^{2} \left[ \sum_{k=1}^{n_{int}} (N_a N_b y J)_{\xi_k, \hat{\theta}_j} W_k \right] \frac{\Delta t}{2} T^\alpha(\hat{\theta}_j) T^\beta(\hat{\theta}_j). 
\]  
(A.3)

Expanding helps elucidate our potential for savings:

\[
\int_Q N_a^\alpha N_b^\beta y dQ = \sum_{k=1}^{n_{int}} (N_a N_b y J)_{\xi_k, \hat{\theta}_1} W_k \left[ \frac{\Delta t}{2} T^\alpha T_1^\beta \right] 
+ \sum_{k=1}^{n_{int}} (N_a N_b y J)_{\xi_k, \hat{\theta}_2} W_k \left[ \frac{T^\alpha T_2^\beta}{2} \right], 
\]  
(A.4)

where \( T_1^\alpha = T^\alpha(\hat{\theta}_1) \) and \( T_2^\beta = T^\beta(\hat{\theta}_2) \).

It is clearer that now that with ESTI, an integral (summation of \( 2 \times n_{int} \)) need only to be computed \( n_{en}^2 \) times rather than \( (n_{en} \times 2)^2 \) times as was done in equation (A.1). Note that in this matrix case, we have cut the number of integrations by a factor of four, whereas in the element-level vector case, we halved the computations. In either case, forming our element-level matrices will still be more costly than forming element-level vectors.

### A.1.1 Derivatives with Respect to Time

Derivatives of shape functions with respect to time will again complicate our formulation. We’ll examine the integral term \( N_a^\alpha N_{b,t}^\beta y \), containing one time derivative. The integration becomes

\[
\int_Q N_a^\alpha N_{b,t}^\beta y dQ = \sum_{j=1}^{2} \sum_{k=1}^{n_{int}} \left( N_a^\alpha N_{b,t}^\beta y \frac{\Delta t}{2} J \right)_{\xi_k, \hat{\theta}_j} W_k, \quad a, b = 1...n_{en}, \quad \alpha = 1, 2. 
\]  
(A.5)

By equation (5.1) we can separate the time and space portions of the shape func-
\[ \int_Q N_a^\alpha N_b^\beta y dQ = \sum_{j=1}^{2} \sum_{k=1}^{n_{int}} \left( N_a T^\alpha \left( (-V \cdot R) \cdot (\nabla_\xi N_b) T^\beta + \frac{2}{\Delta t} N_b T^\beta_{t,\theta} \right) y \frac{\Delta t}{2} J \right)_{\xi_k,\tilde{\theta}_j} W_k. \]

Because \( T^\alpha(\tilde{\theta}_j) T^\beta(\tilde{\theta}_j) \) and \( T^\alpha(\tilde{\theta}_j) T^\beta_{t,\theta}(\tilde{\theta}_j) \) are independent of \( \tilde{\xi}_k \), we can take them out of the \( k \) summation:

\[ \int_Q N_a^\alpha N_b^\beta y dQ = \sum_{j=1}^{2} \left[ \sum_{k=1}^{n_{int}} (N_a (-V \cdot R) \cdot (\nabla_\xi N_b) y J)_{\xi_k,\tilde{\theta}_j} W_k \right] \frac{\Delta t}{2} T^\alpha(\tilde{\theta}_j) T^\beta(\tilde{\theta}_j) + \sum_{j=1}^{2} \left[ \sum_{k=1}^{n_{int}} (N_a N_b y J)_{\xi_k,\tilde{\theta}_j} W_k \right] \frac{\Delta t}{2} T^\alpha(\tilde{\theta}_j) \frac{2}{\Delta t} T^\beta_{t,\theta}(\tilde{\theta}_j). \]

We already know the values of \( T^\alpha(\theta_j) \) from equations (5.4) and (5.5), so once we substitute \( j = 1, 2 \) we're left with an expanded equation comparable to equation (A.4):

\[ \int_Q N_a^\alpha N_b^\beta y dQ = \sum_{k=1}^{n_{int}} (N_a (-V \cdot R) \cdot (\nabla_\xi N_b) y J)_{\xi_k,\tilde{\theta}_1} W_k \left[ \frac{\Delta t}{2} T^\alpha_{11} T^\beta_{11} \right] + \sum_{k=1}^{n_{int}} (N_a N_b y J)_{\xi_k,\tilde{\theta}_1} W_k \left[ \frac{(-1)^\beta}{2} T^\alpha_{1} \right] + \sum_{k=1}^{n_{int}} (N_a (-V \cdot R) \cdot (\nabla_\xi N_b) y J)_{\xi_k,\tilde{\theta}_2} W_k \left[ \frac{\Delta t}{2} T^\alpha_{22} T^\beta_{22} \right] + \sum_{k=1}^{n_{int}} (N_a N_b y J)_{\xi_k,\tilde{\theta}_2} W_k \left[ \frac{(-1)^\beta}{2} T^\alpha_{2} \right]. \]

### A.2 SSTF Implementation

Implementation of element-level matrices, is again, fairly similar to the element-level vector case. We will turn back to our previous example of a generic integral for an element-level vector from equation (A.3). The formulation for a typical element-level
matrix will look similar to this:

\[
\int_Q N_a^\alpha N_b^\beta y dQ = \sum_{j=1}^{n_{\text{init}}} \left[ \begin{array}{c}
\left( N_a^1 N_b^1 y \frac{\Delta t}{2} J \right) | \hat{\xi}_{k, \delta_j} W_k \\
\left( N_a^1 N_b^2 y \frac{\Delta t}{2} J \right) | \hat{\xi}_{k, \delta_j} W_k \\
\left( N_a^2 N_b^1 y \frac{\Delta t}{2} J \right) | \hat{\xi}_{k, \delta_j} W_k \\
\left( N_a^2 N_b^2 y \frac{\Delta t}{2} J \right) | \hat{\xi}_{k, \delta_j} W_k 
\end{array} \right].
\]  

(A.9)

Note, the matrix size is \(2 \times n_{\text{en}} \times 2 \times n_{\text{en}}\). Whereas \(Y\) was a vector in equation (5.41), we will define it as a matrix to assist us with forming our element-level vector. Notice that our new vector \(Y\) is of size \(n_{\text{en}} \times 2 \times n_{\text{en}}\) (each \(Y_j\) is of size \(n_{\text{en}} \times n_{\text{en}}\)). Instead of summing \(j = 1, 2\), we will store in different parts of the vector \(j = 1\) and \(j = 2\):

\[
Y = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix}
\sum_{k=1}^{n_{\text{init}}} (N_a N_b y J) | \hat{\xi}_{k, \delta_1} W_k \\
\sum_{k=1}^{n_{\text{init}}} (N_a N_b y J) | \hat{\xi}_{k, \delta_2} W_k
\end{bmatrix}.
\]  

(A.10)

To complete our implementation, we must recompile our \(Y\) vectors and finally come back to calculate our integral to get a solution identical to equation (A.9):

\[
\int_Q N_a^\alpha N_b^\beta y dQ = \begin{bmatrix}
Y_1^{\Delta t} T_1^1 T_1^1 + Y_2^{\Delta t} T_1^1 T_2^1 \\
Y_1^{\Delta t} T_2^1 T_1^1 + Y_2^{\Delta t} T_2^1 T_2^1
\end{bmatrix}.
\]  

(A.11)

which is a \(2 \times n_{\text{en}} \times 2 \times n_{\text{en}}\) sized matrix.

The implementation changes very little for integrals containing derivatives of shape functions with respect to space.
Bibliography


[15] T. Tezduyar, M. Behr, and J. Liou. A new strategy for finite element computations involving moving boundaries and interfaces – the deforming-spatial-


