University of Cape Town

Masters Dissertation

Mesh Adaptation through r-Refinement using a Truss Network Analogy

Author:
Bevan W.S. Jones

Supervisor:
Prof. Arnaud G. Malan

Quote

"Take the risk of thinking for yourself; much more happiness, truth, beauty, and wisdom will come to you that way."

- Christopher Hitchens

Acknowledgements

- To my parents, Nadia and Haydn Jones

On numerous occasions during my mischievous childhood I would enquire as to why parents punished their children. Always the response was the same, "When you grow up, you will thank me." I now feel I understand their wise reply, and while I do not believe that any child will ever be able to express all their gratitude for loving parents, I do hope that this work is in some way a meaningful demonstration of my never ending appreciation.

For everything, mom and dad, thank you.

- To my supervisor, Arnaud Malan

When I met Arnaud for the first time I was in a bad place in my life, from personal trouble to a failed first attempt at a masters. In difficult times and times of self doubt two things seem, to me at least, to be of importance; A purpose and confidence. The supervision of this project has been more than academic and has given me both. My personal growth, guided by Arnaud's wisdom, has been invaluable over these past years.

I am truly grateful for your guidance in all spheres

- To my research group

The people who make up the CFD industrial research group and others with whom we share space have been exceedingly helpful over the course of this project. Many of whom have offered hours of their own time to help get me through. I have made some good friends along the way and learned much about working in teams.

Thank you for all the help, it is much appreciated

UCT ICTS HPC Cluster
Numerous computations were performed using facilities provided by the University of Cape Town's ICTS High Performance Computing team: http://hpc.uct.ac.za

Abstract

This project investigates the use of a truss network, a structural mechanics model, as a metaphor for adapting a computational fluid dynamics (CFD) mesh. The objective of such adaptation is to increase computational efficiency by reducing the numerical error. To drive the adaptation, or to give the scheme an understanding of accuracy, computational errors are translated into forces at mesh vertices via a so-called monitor function. The ball-vertex truss network method is employed as it offers robustness and is applicable to problems in both two and three dimensions. In support of establishing a state-of-the-art adaptive meshing tool, boundary vertices are allowed to slide along geometric boundaries in an automated manner. This is achieved via feature identification followed by the construction of 3^{rd} order bezier surface patches over boundary faces. To investigate the ability of the scheme, three numerical test cases were investigated. The first comprised an analytical case, with the aim of qualitatively assessing the ability to cluster vertices according to gradient. The developed scheme proved successful in doing this. Next, compressible transonic flow cases were considered in 2D and 3D. In both cases, the computed coefficient of lift and moment were investigated on the unrefined and refined meshes and then compared for error reduction. Improvements in accuracy of at least 60% were guaranteed, even on coarse meshes. This is viewed as a marked achievement in the sphere of robust and industrially viable r-refinement schemes.

Contents

List of Figures

List of Tables

Nomenclature

Greek Symbols

β	Angle between two boundary edges	
ϵ	Richardson extrapolation variable difference	
η_{sf}	Safety factor	
γ	Frequency factor for whirl test case	
Λ	Aerofoil taper ratio	
λ	Face group index	
ω	Relaxation factor	
ϕ	Flow variable for monitor function	
ρ	Density	$kg.m^{-3}$
σ	Matrix entry	
$\varkappa$	Curvature	
ϑ	Variable difference ratio	
ξ	Ball-vertex coefficient 2D	
ζ	Richardson extrapolation variable	
η	Ball-vertex coefficient 3D	
θ	Angle between vectors	rad

Mathematical Operators

$'$	Prime

$''$	Double prime	
$\cdot$	Vector dot product	
Δ	Vector component change in	
δ	Change in	
$\frac{d}{dx}$	Differential operator	
$\hat{\boldsymbol{e}}$	Unit vectors	
$\sum$	Sum	
$\times$	Vector cross product	
$\vec{t}$	General unit vector	
^{T}	Matrix transpose	

Roman Symbols

$\bar{\boldsymbol{K}}$	System Stiffness Matrix	
$\bar{\boldsymbol{T}}$	Unit vector matrix	
$\delta\vec{x}$	Displacement of a node	m
$\vec{G}$	Gregory patch control points	
$\vec{N}$	Boundary node normal	
$\vec{n}$	Boundary face normal	
$\vec{P}$	Bèzier surface degree elevated control points	
$\vec{V}$	Bèzier curve control points	
$\vec{W}$	Bèzier curve vectors	
$\vec{x}$	Position vector of a node	
A	Area (surface or cross-sectional)	m^2
a	Cubic curve coefficient	
B	Bernstein basis polynomials	
C	Force coefficient	
c_l	Coefficient of lift	

c_m	Coefficient of moment	
D	Bèzier curve polynomial	
d	Dimension of the simulation	
E	Young's modulus	Pa
e	Edge index	
F	External force	N
f	Internal member force	N
g	Parametric cubic spline	
GCI	Grid Convergence Index	
i	Node index	
j	Connected node index	
k	Spring constant	$N.m^{-1}$
L	Length	m
l	Parameter to parametric curves	
l_{ch}	Characteristic chord length	m
LHS	Left Hand Side	
p	Pseudo node or polynomial/accuracy order	
P_{dyn}	Dynamic pressure	Pa
q	Point on a Bèzier curve	
r	Grid refinement ratio	
RHS	Right Hand Side	
u	Element index or boundary face index	
w	Barycentric co-ordinates	
x, y, z	Co-ordinates in principle direction	

Subscripts and Superscripts

^{i,j}	Iteration number

$_b$	Bèzier curve/surface point locator indices
$_c$	Curvature
$_g$	Gradient
$_m$	Maximum
$_n$	Normal
$_t$	Tangent
$_{cl}$	Collapse
$_{pj}$	Projection
x, y, z	Vector component in principle directions

1 Introduction

1.1 Project Background

1.1.1 The Need for Mesh Adaptation

The use of computational fluid dynamics (CFD) to solve multi-physics problems has seen exponential growth over the past two decades. As computational power has increased, so the number of elements in a typical simulation has also increased. This has resulted in continued improvements in CFD accuracy. Despite this, computational cost remains a major challenge and is expressed through the limitation of available hardware for simulations. Foremost is the accuracy of the simulation and here, the higher the desired fidelity, the higher the hardware requirements which, in turn, increase cost. This includes both man hours (for example, generating refined meshes) as well as that due to computer hardware and electricity.

Clearly the improvement of computational efficiency, i.e. accuracy vs. computational cost, should remain an active area of research. Mesh adaptation is a prime example which involves placing vertexes so as to reduce errors arising from the underlying discretization process. Areas of the flow with high gradients and curvatures or where mesh spacing is inappropriately large are typically associated with high errors. This significantly worsens efficiency of a field solver. Therefore, by improving the error topology over a mesh, a mesh adaptation process can be employed to achieve greater computational efficiency.

The effect here then, is to increase computational efficiency via reducing error through relocating nodes to areas of high field variable gradient and/or curvature (which implies discretization error). However, the degree to which local element spacing can be manipulated via manual re-meshing is limited to what is known about the flow field *a priori*. Additionally, even if the flow field is well understood, the placement of the nodes into areas of predicted high gradient is not exact if done manually. This is ascribed to flow phenomena, such as shocks and vortices, being highly irregular and often taking convoluted and complex patterns. The resulting mesh, while better, is still typically sub-optimal and costly in the sense that repeated meshing and CFD calculations are still required.

Mesh adaptation may also be effected in a more automated fashion during the solution process which offers *a posteriori* knowledge of field. For such adaptation methods, robustness via ensuring a valid mesh is critical, while applicability to complex geometries (in 2D and 3D) is key to ensure industrial relevance. A number of approaches have been developed to utilise known flow features and produce higher fidelity results, namely p, h, and r refinement. While 'r' is the subject of this work, the others are discussed briefly so as to give background context.

1.1.2 p-Refinement

p-Refinement adapts the error topology by improving the discretization accuracy in local areas of the mesh, thereby improving regional accuracy. Apart from isolated examples, this method has not yet shown significant potential to improve computational efficiency for routine compressible and incompressible flow simulations as compared to conventional 2^{nd} order accurate methods [2]. In passing, it is noted that p-Refinement has previously been coupled with r-refinement schemes in the arena of contact problems [3]; this work is, however, focussed on r-refinement applied to CFD.

1.1.3 h-Refinement

Perhaps the most widely used scheme commercially; h-refinement involves the insertion and deletion of mesh vertices with the goal to increase node density in high error regions and, conversely, removing nodes in low error regions. Clearly, h-adaptation operates on mesh connectivity. Modern h-refinement can trace its roots back to the late 70's and early 80's [4, 5] and has continued to grow in industrial maturity. Aftosmis et al. [6] demonstrated this with applications to aerofoils, space craft and rockets on cut-cell cartesian, or structured meshes. Additionally, h-refinement was applied successfully to transient problems such as blast wave modelling by Alauzet et al. [7].

Perhaps the reason for its popularity is the ability to take a coarse initial mesh and obtain a solution of almost any desired accuracy (given adequate computational resource). This feature of h-schemes lies in the ability to continuously add nodes until the mesh has sufficient nodes such that the flow equations produce an acceptably accurate solution. The clear robustness of this method is offset by the significant increase in complexity of the mesh. Additionally, in the case of parallel computations, the mesh would need to be re-decomposed so that unbalanced CPU core workload is re-balanced. Due to such concerns, it is starting to emerge that it is better suited to be used in conjunction with other refinement schemes, such as r and p types [8, 9]. By coupling the schemes certain synergies can be realised. As an example, r-refinement can be used to optimise a certain mesh without any impact on

connectivity. If solution accuracy is still insufficient, a limited amount of h-adaptivity can be introduced.

1.1.4 r-Refinement

As alluded to previously, r-refinement, a cousin of h-refinement, seeks to move nodes for one of two purposes. Either to deform a mesh boundary (typical to fluid-structure interaction (FSI) problems), or to reduce maximum error in the mesh. The latter is typically done via relocating and clustering nodes in areas where the flow field variable's gradients and/or curvatures are high. Note that it is typically assumed in CFD that these high curvatures and gradients are typically the area of high error. The difference between h- and r-adaptivity is that connectivities are not altered. The underlying principle is thus to achieve an equi-distribution of the error over the mesh, ideally resulting in the optimal mesh for a given number of nodes. One of the first to exploit this concept was de Boor [10].

r-Refinement is still in its infancy when compared with h- and p- types of refinement, which are now of industrial strength [11]. Its behaviour is less well documented and further work is still required. Much of the difficulty with r-refinement is the robustness of the schemes developed to date. Mesh entanglement, in which a mesh suffers negative volume elements post adaptation or unacceptable element quality results are still of concern.

r-Refinement, sometimes refereed to as Moving Mesh Methods (MMM), can be broken into two broad groups viz. location based and velocity based methods. Velocity based methods drive mesh adaptation by solving for vertex velocities. Location based methods, by comparison, adapt by determining the density or position of nodes by solving some mapping functional between a computational and physical domain. What follows is a brief survey of the r-refinement literature compiled primarily from three sources viz. Tang [12], Budd, Huang and Russell [11] and Baines, Hubbard and Jimach [13]. Readers seeking more in-depth information should consult the aforementioned cited works.

Velocity Based MMM, sometimes referred to as Lagrangian or Arbitrary Lagrangian Eulerian methods, calculate a velocity at mesh points and then extract the displacement for the nodes at some instantaneous time [13]. Broadly, these methods fall into two main categories: those which employ some physical metaphor [14–16] and those that take a mathematical approach [17–19]. Since the velocity based algorithms tend to be applied to moving boundary problems more than to error minimisation, further elaboration is deemed unwarranted.

Location Based MMM may be classified into variational and optimal transport type schemes. In the case of the aforementioned, a so-called transform functional may be employed as the monitor function [20,21]. The latter serves to minimize error while retaining mesh quality. Further simplifications

to the discretization of the method were done by Ceniceros and Hon [22]. An extension of the variational approach involved the use of a harmonic mapping functional by Dvinsky [23] and Brackbill and Satzman [24], which lead to the formulation of Euler-Lagrange equations used to control mesh concentration and quality [25, 26].

By comparison, optimal transport methods (the second class of location based MMM) seek to equidistribute the least-squares norm of some monitor function over the mesh. The resulting minimisation problem is the differential geometry Monge-Kantorovich problem. The existence of a unique solution was proven by Brenier [27] or Caffarelli [28, 29] and later more specifically for adaptive meshes by Delzanno [30]. The mapping function is thus unique in the case of a convex domain as a result of the proof which yields a Monge-Ampére equation. The method enjoyed some popularity and has been demonstrated in 3D [31] and used on real world problems, such as weather modelling [32], yet the domains presented remain relatively simple.

1.1.4.1 Truss networks

A common thread through the outlined r-refinement techniques is the duality in their ability to solve for moving boundary problems (mesh deformation) as well as for mesh error minimisation. The mechanics of both methods utilise some monitor function to drive the adaptive process, whether it be for boundary related changes or the desire to change the error topology over the mesh. Indeed, some methods are better suited to certain problems, but the fact remains that an r-refinement algorithm should have this duality in nature. A scheme's ability to avoid negative volume elements while being applicable to complex geometries is of importance. Here, a truss network approach, wherein spring stiffness is inversely proportional to edge length, reduces such negative volume creation.

This leads to the consideration of the truss network or spring analogy as a mesh movement method for CFD meshes. Batina [33] was one of the first to do so for moving boundary problems. In its simplest form each edge in the mesh becomes a fictitious spring. Blom [34] furthered the work for truss networks by generalising the two main approaches to truss networks for CFD applications. In the first, the vertex spring assumes that the equilibrium length is zero, while the second employs the initial lengths of the springs as done by Batina [33]. With that said, however, stability of truss networks is dependant on triangles and CFD mesh elements are far more diverse. Thus, augmentations to the standard truss have been devised to improve scheme robustness and stability.

The torsional method, pioneered by Farhat et al. [35], attaches a torsional spring to each vertex in each attached element. The stiffness for these springs is chosen such that, as the angle between the edges (element faces) connected to the vertex approaches 0 or π, the stiffness tends to infinity. The method

was later extended to three dimensions [36] and confirmed by Burg [37]. *The semi-torsional method* proposed by Blom [34] is a variation on the torsional method, which employs the angle opposite an edge spring (in a triangle for example) to modify the stiffness; thus eliminating the need for a torsional spring altogether. Extension into three dimensions was done by Zeng and Either [38]. The *ortho-semi-torsional method*, devised by Markou et al. [39], extends the semi-torsional method with a blend of the so-called ball-vertex method.

The ball-vertex method was first proposed by Bottasso [1, 40]. This method involves the addition of a spring between each vertex of the element and the opposite element face (the face through which the vertex would pass if the element were to become concave). The added spring is placed orthogonal to this face. Compared to the torsional spring method, the ball-vertex variant is reported to offer improved robustness under severe deformations [1].

Recent work on improvement on the ball-vertex method has been done by Lin et al. [41], in which the total global stiffness matrices are broken down into small systems of one node and its connected nodes. Each of these subsystems is then solved to obtain the new position of the node and updated once all new vertex positions have been calculated, in much the same way as a smoothing algorithm. The above cited works on truss networks typically involve mesh deformation problems (such as FSI). However, the ability to coerce these methods for error reduction is, in addition to being plausible, exciting due to both novelty and industrial relevance to complex geometries.

1.2 Project Scope

Currently, the UCT industrial CFD research group's primary area of expertise lies in the solver area. However the demands of modern CFD require a more holistic approach, and the underlying meshes need to be explored as areas for improvement in accuracy and speed. The purpose of this project is to set out the base work for adaptive meshing with the objective of demonstrating improved accuracy and good scheme robustness when applied to industrial problems (such as aerofoils under transonic flow). Understanding that the ultimate adaptive meshing solution will most likely involve a combination p-, h- and r-refinement, this project will focus on r-.

From the literature surveyed, it seems apparent that the majority of r-refinement papers focus on rather simple geometries. In fact, the majority of test cases performed were on square domains where some analytical function was applied. In addition, the scheme's desirable effects are only reliable under certain conditions. An example is the optimal transport method where domains must be convex in order to obtain the unique solution of the Monge-Ampére equation.

In an attempt to circumvent the above, it was observed that truss-networks have demonstrated robustness and applicability to complex geometries for moving boundary problems. Extending such to error reduction, therefore, appears a valid proposition. Inspired by the Ph.D. thesis of Acikgoz [42], it was opted to consider the ball-vertex method for this study. Though the focus of her work was more on h-adaptivity and mesh deformation for FSI problems, the closing chapter briefly explores application to error field reduction. The problem considered involved a 2D shock problem. Though little was done by way of quantifying increase in accuracy, potential of the the method for error reduction was demonstrated.

The implementation here differs from Acikgoz [42] in a number of ways. These include improving the approximation of stiffness change due to truss rotations and the formulation of a simple, yet novel monitor function for complex CFD applications. This work has the additional goal of demonstrating applicability of the ball-vertex method to industrial problems, with the aim of error minimisation. In order to demonstrate industrial relevance, the developed technology is applied to transonic flow on aerofoils in both 2D and 3D.

1.3 Plan of Development

Following this introduction, Chapter 2 details the classic ball-vertex method. The loading of the system is considered with the aim of obtaining the metrics by which the mesh adaptation is driven. Also discussed are some peripheral components of the scheme.

Chapter 3 focuses on the handling of the boundary nodes and the modifications to the governing system of equations. Details of boundary identification algorithms are given. This is used in conjunction with the technique developed to ensure that boundary nodes remain on the boundary surface, even without an explicit definition of such.

Chapter 4 describes the integration of the developed r-refinement technology into the Elemental^{TM} CFD software. In addition, the process followed to achieve node movement in an efficient and robust manner is outlined.

Numerical results are then presented in Chapter 5, starting with some analytical test cases in 2D with analytical fields. This demonstrates algorithm robustness and ability to refine for given error topologies. The remainder of the chapter is devoted to application of the method to a challenging and industrially relevant CFD problem viz. transonic inviscid flow over an aerofoil. In the interest of a rigorous evaluation, coarse and fine meshes are considered in both 2D and 3D. Finally, a mesh deformation in 3D is attempted before Chapter 6 concludes the dissertation.

2 A Truss Network Approach to r-Refinement

2.1 Truss Networks

A brief re-derivation of truss networks adapted from Bucciarelli [43] follows. Given a solid truss structure, it is possible to calculate the displacements of vertices given an applied load. A solution can be obtained by taking into account member stiffnesses, which are a function of their geometry and material properties. Thus member displacement will be weighted towards (materially) softer members; mechanically, this is the uptake of elastic potential energy for a member, given an applied load.

The uptake of energy due to a displacement is analogous to the mechanics of springs. From Hook's Law it is known that the resistance force can be expressed as

$$k\delta\vec{L} = \vec{f} \tag{2.1}$$

where f is the spring force, k the spring stiffness and δL the change of length. Given a truss structure, such as that in Figure 2.1, a force balance at the vertices can be obtained as:

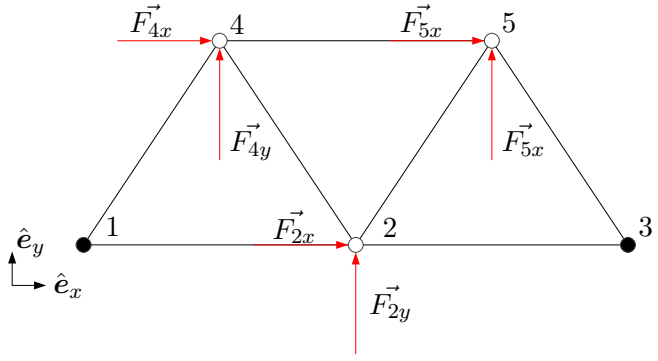

Fig. 2.1: General truss network structure constrained at nodes 1 and 3

For equilibrium at each node:

$$\begin{aligned}
\sum f_{2x} &: -f_{12}\hat{\boldsymbol{e}}_x + f_{23}\hat{\boldsymbol{e}}_x + f_{24}\hat{\boldsymbol{e}}_x + f_{25}\hat{\boldsymbol{e}}_x = \vec{F}_{2x} \\
\sum f_{2y} &: -f_{12}\hat{\boldsymbol{e}}_y + f_{23}\hat{\boldsymbol{e}}_y + f_{24}\hat{\boldsymbol{e}}_y + f_{25}\hat{\boldsymbol{e}}_y = \vec{F}_{2y} \\
\sum f_{4x} &: -f_{14}\hat{\boldsymbol{e}}_x - f_{24}\hat{\boldsymbol{e}}_x + f_{45}\hat{\boldsymbol{e}}_x = \vec{F}_{4x} \\
\sum f_{4y} &: -f_{14}\hat{\boldsymbol{e}}_y - f_{24}\hat{\boldsymbol{e}}_y + f_{45}\hat{\boldsymbol{e}}_y = \vec{F}_{4y} \\
\sum f_{5x} &: -f_{25}\hat{\boldsymbol{e}}_x - f_{35}\hat{\boldsymbol{e}}_x - f_{45}\hat{\boldsymbol{e}}_x = \vec{F}_{5x} \\
\sum f_{5y} &: -f_{25}\hat{\boldsymbol{e}}_y - f_{35}\hat{\boldsymbol{e}}_y - f_{45}\hat{\boldsymbol{e}}_y = \vec{F}_{5y}
\end{aligned}$$

where numeric subscripts denote vertex numbers and principle co-ordinate directions (x or y). Further, f and F are truss internally and externally applied forces respectively. Note that the above formulation does not hold for large displacements in terms of significant alterations in length and orientation (due to rotation). The above system can be represented with a matrix $\bar{\boldsymbol{T}}$ of unit vectors $\vec{t}$, where each unit vector is aligned along the truss, as

$$\begin{pmatrix}
-\vec{t}_{12x} & 0 & \vec{t}_{23x} & \vec{t}_{24x} & \vec{t}_{25x} & 0 & 0 \\
-\vec{t}_{12y} & 0 & \vec{t}_{23y} & \vec{t}_{24y} & \vec{t}_{25y} & 0 & 0 \\
0 & -\vec{t}_{14x} & 0 & -\vec{t}_{24x} & 0 & 0 & \vec{t}_{45x} \\
0 & -\vec{t}_{14y} & 0 & -\vec{t}_{24y} & 0 & 0 & \vec{t}_{45y} \\
0 & 0 & 0 & 0 & -\vec{t}_{25x} & -\vec{t}_{35x} & -\vec{t}_{45x} \\
0 & 0 & 0 & 0 & -\vec{t}_{25y} & -\vec{t}_{35y} & -\vec{t}_{45y}
\end{pmatrix}
\begin{pmatrix} f_{12} \\ f_{14} \\ f_{23} \\ f_{24} \\ f_{25} \\ f_{35} \\ f_{45} \end{pmatrix}
=
\begin{pmatrix} \vec{F}_{2x} \\ \vec{F}_{2y} \\ \vec{F}_{4x} \\ \vec{F}_{4y} \\ \vec{F}_{5x} \\ \vec{F}_{5y} \end{pmatrix} \tag{2.2}$$

where the relationship for $\vec{f}$ is given by Equation (2.1). In traditional solid mechanics, the spring constant k would be a function of material properties and member geometry, namely $\frac{AE}{L}$ (with A and E respectively the area and Young's modulus). However, when considering r-refinement, this physical meaning is lost. A new stiffness is formulated via edge length and is discussed in Section 2.3. Next, the formulation of the relationship between the truss member length $\vec{f}$ and the nodal displacements are considered.

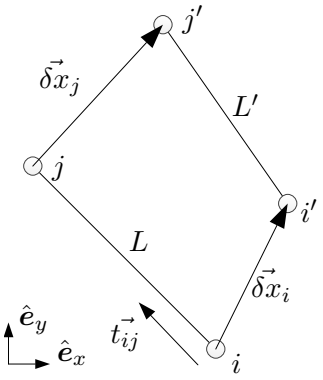

Fig. 2.2: Truss member displacement diagram

The truss member's change in length from L to L' can be expressed using the displacement $\vec{\delta x}$ vectors depicted in Figure 2.2. Assuming negligible rotation, the change in length may be computed as

$$\delta L = L' - L \approx \vec{\delta x}_j \cdot \vec{t}_{ij} - \vec{\delta x}_i \cdot \vec{t}_{ij} \tag{2.3}$$

where subscripts i and j denote the nodes in Figure 2.2. Applying Equation (2.3) to the truss network in Figure 2.1 yields

$$\begin{pmatrix} \vec{\delta L}_{12} \\ \vec{\delta L}_{14} \\ \vec{\delta L}_{23} \\ \vec{\delta L}_{24} \\ \vec{\delta L}_{25} \\ \vec{\delta L}_{35} \\ \vec{\delta L}_{45} \end{pmatrix} = \begin{pmatrix} -\vec{t_{12x}} & -\vec{t_{12y}} & 0 & 0 & 0 & 0 \\ 0 & 0 & -\vec{t_{14x}} & -\vec{t_{14y}} & 0 & 0 \\ \vec{t_{23x}} & \vec{t_{23y}} & 0 & 0 & 0 & 0 \\ \vec{t_{24x}} & \vec{t_{24y}} & -\vec{t_{24x}} & -\vec{t_{24y}} & 0 & 0 \\ \vec{t_{25x}} & \vec{t_{25y}} & 0 & 0 & -\vec{t_{25x}} & -\vec{t_{25y}} \\ 0 & 0 & 0 & 0 & -\vec{t_{35x}} & -\vec{t_{35y}} \\ 0 & 0 & \vec{t_{45x}} & \vec{t_{45y}} & -\vec{t_{45x}} & -\vec{t_{45y}} \end{pmatrix} \begin{pmatrix} \Delta x_2 \\ \Delta y_2 \\ \Delta x_4 \\ \Delta y_4 \\ \Delta x_5 \\ \Delta y_5 \end{pmatrix} \tag{2.4}$$

Here, Δx and Δy denote the change in node co-ordinates. Substituting Equations (2.4) and (2.6) into Equation (2.1) yields an expression for $\vec{f}$, which can be further substituted into Equation (2.2). This yields the prototype governing equation of the scheme

$$\begin{aligned} \bar{\boldsymbol{T}}\vec{f} &= \vec{F} \\ \bar{\boldsymbol{T}}\bar{\boldsymbol{k}}\bar{\boldsymbol{T}}^T\vec{\Delta x} &= \vec{F} \\ \bar{\boldsymbol{K}}\vec{\Delta x} &= \vec{F} \end{aligned} \tag{2.5}$$

where $\bar{\boldsymbol{k}}$ is the member stiffness matrix of the form

$$\bar{\boldsymbol{k}} = \begin{pmatrix} k_{12} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & k_{14} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & k_{23} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & k_{24} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & k_{25} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & k_{35} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & k_{45} \end{pmatrix} \tag{2.6}$$

and $\bar{\boldsymbol{K}}$ represents the well known system stiffness matrix. In expanded form, the above matrix equation (Equation (2.5)) may be represented as

$$\begin{pmatrix} \sigma_{e,xx} & \sigma_{e,xy} & \sigma_{24,xx} & \sigma_{24,xy} & \sigma_{25,xx} & \sigma_{25,xy} \\ & \sigma_{e,yy} & \sigma_{24,yx} & \sigma_{24,yy} & \sigma_{25,yx} & \sigma_{25,yy} \\ & & \sigma_{e,xx} & \sigma_{e,xy} & \sigma_{45,xx} & \sigma_{45,xy} \\ & & & \sigma_{e,yy} & \sigma_{45,yx} & \sigma_{45,yy} \\ & Symm & & & \sigma_{e,xx} & \sigma_{e,xy} \\ & & & & & \sigma_{e,yy} \end{pmatrix} \begin{pmatrix} \Delta x_2 \\ \Delta y_2 \\ \Delta x_4 \\ \Delta y_4 \\ \Delta x_5 \\ \Delta y_5 \end{pmatrix} = \begin{pmatrix} F_{2x} \\ F_{2y} \\ F_{4x} \\ F_{4y} \\ F_{5x} \\ F_{5y} \end{pmatrix} \tag{2.7}$$

For off diagonals' entries, $\sigma_{24,xx}$ for example, denotes the matrix entry constructed from data obtained from the edge connecting nodes 2 and 4 for the principle co-ordinate directions xx. The main diagonal and selected off diagonal entries (see condition in Equation (2.8)) are a summation of all the data from the edges e, connected to the node. A matrix entry can thus be expressed as

$$\begin{aligned} \sigma_{e,d_1 d_2} &= \begin{cases} \sum_e \left| k_e \left(\vec{t}_e \cdot \hat{\boldsymbol{e}}_{d_1}\right) \left(\vec{t}_e \cdot \hat{\boldsymbol{e}}_{d_2}\right) \right| & if \ \frac{row}{col} = i \times d_m + 0/1/(2) \\ -\left| k_e \left(\vec{t}_e \cdot \hat{\boldsymbol{e}}_{d_1}\right) \left(\vec{t}_e \cdot \hat{\boldsymbol{e}}_{d_2}\right) \right| & else \end{cases} \\ e &= [0 : e_m]; d_1, d_2 \in [x, y, (z)] \end{aligned} \tag{2.8}$$

where d_1 and d_2 indicate which combination of principle unit vectors are used. In the above, i denotes the node index and d_m the solution dimension. The vector on the RHS of Equation (2.7) is refereed further as the load vector $\vec{F}$. Clearly, for larger truss systems, $\bar{\boldsymbol{K}}$ becomes a sparse symmetric matrix.

2.2 Ball-Vertex Method

As noted previously, in Chapter 1, a truss structure is only structurally stable for triangulated shapes. Mesh elements have a much wider vocabulary of shapes; thus, an augmentation to a standard truss system is required. The ball-vertex method, although not the only method [34–39], provides this augmentation. The core derivations of this method are taken from [1, 40].

Firstly, the two dimensional case is formulated and then extended for three dimensions. For consistency, the springs which arise due to the ball-vertex method are referred to as element springs.

2.2.1 2D Formulation

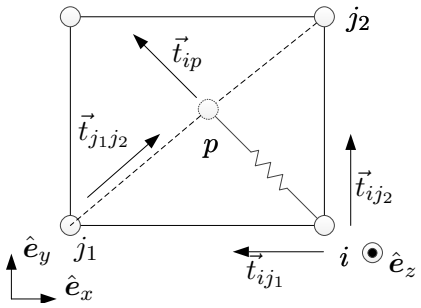

Fig. 2.3: The Ball-Vertex Method applied in 2 dimensions.

The additional diagonal spring added by the ball-vertex method is shown in Figure 2.3. First, the co-ordinates for the pseudo node p are obtained by calculation of the length L_{ip} between nodes i and p:

$$\begin{aligned} \vec{t}_{ip} &= \hat{\boldsymbol{e}}_z \times \vec{t}_{j_1 j_2} \\ \textbf{if } \vec{t}_{ip} \cdot \vec{t}_{ij_1} < 0 &\rightarrow \vec{t}_{ip} = -\vec{t}_{ip} \\ L_{ip} &= \vec{t}_{ip} \cdot L_{ij_1} \vec{t}_{ij_1} \\ \vec{x}_p &= \vec{x}_i + L_{ip} \vec{t}_{ip} \end{aligned} \tag{2.9}$$

The addition of nodes (degrees of freedom) is circumvented in an elegant manner via writing the displacements of the pseudo nodes in terms of the nodes which describe the lines (2D), or plane (3D) on which it resides. Thus, a solution for the displacement of the pseudo nodes in terms of the surrounding nodes (nodes j_1 and j_2 in the Figure 2.3) can be obtained by interpolation as

$$\vec{x}_p = \xi \vec{x}_{j_1} + (1 - \xi) \vec{x}_{j_2} \tag{2.10}$$

where $\vec{x}$ denotes nodal co-ordinates. Solving for the interpolation coefficient ξ,

$$\xi = \frac{(\vec{x}_{j_1} - \vec{x}_{j_2}) \cdot (\vec{x}_p - \vec{x}_{j_1})}{|\vec{x}_{j_1} - \vec{x}_{j_2}|} \tag{2.11}$$

the displacement for node p ($\delta\vec{x}_p$) may be represented as

$$\delta\vec{x}_p = \xi\delta\vec{x}_{j_1} + (1-\xi)\delta\vec{x}_{j_2} \tag{2.12}$$

The stiffness contribution for node i into the stiffness matrix for the element in Figure 2.3 can now be added as follows

$$\sigma_{ip,d_1d_2} = \left| k_{ip} \left(\vec{t}_{ip} \cdot \hat{\boldsymbol{e}}_{d_1}\right) \left(\vec{t}_{ip} \cdot \hat{\boldsymbol{e}}_{d_2}\right) \right|$$

$$d_1, d_2 \in [x, y]$$

$$\begin{pmatrix} \sigma_{ip,xx} & \sigma_{ip,xy} & -\xi\sigma_{ip,xx} & -\xi\sigma_{ip,xy} & (\xi-1)\sigma_{ip,xx} & (\xi-1)\sigma_{ip,xy} \\ & \sigma_{ip,yy} & -\xi\sigma_{ip,yx} & -\xi\sigma_{ip,yy} & (\xi-1)|\sigma_{ip,yx} & (\xi-1)\sigma_{ip,yy} \\ & & \xi^2\sigma_{ip,xx} & \xi^2\sigma_{ip,xy} & \xi(1-\xi)\sigma_{ip,xx} & \xi(1-\xi)\sigma_{ip,xy} \\ & & & \xi^2\sigma_{ip,yy} & \xi(1-\xi)\sigma_{ip,yx} & \xi(1-\xi)\sigma_{ip,yy} \\ & & Symm & & (1-\xi)^2\sigma_{ip,xx} & (1-\xi)^2\sigma_{ip,xy} \\ & & & & & (1-\xi)^2\sigma_{ip,yy} \end{pmatrix} \tag{2.13}$$

with nomenclature previously defined.

2.2.2 Extension to 3D

Extension of the above methodology to 3D is straightforward. Considering the hexahedron in Figure 2.4, it follows that

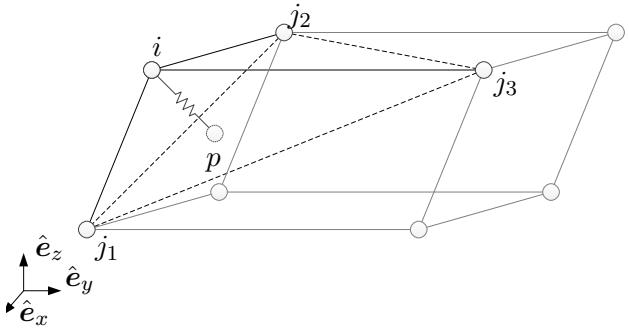

Fig. 2.4: Implementation of a ball-vertex spring in 3 dimensions [1].

$$\vec{t}_{ip} = \vec{t}_{j_1 j_2} \times \vec{t}_{j_1 j_3} \tag{2.14}$$

The ratios used in the stiffness matrix must be determined, noting that in 3D, node p's position is described by the three nodes which are co-planar

$$\vec{x}_p = \xi \vec{x}_{j_1} + \eta \vec{x}_{j_2} + (1 - \xi - \eta)\vec{x}_{j_3} \tag{2.15}$$

Here, the second interpolation coefficient η is given by

$$\eta = \frac{(\vec{x_{j_1}} - \vec{x_{j_3}}) \cdot (\vec{x_p} - \vec{x_{j_1}})}{|\vec{x_{j_1}} - \vec{x_{j_3}}|} \tag{2.16}$$

and ξ follows from Equation (2.11). The interpolated displacements then follow as

$$\vec{\delta x}_p = \xi \vec{\delta x}_{j_1} + \eta \vec{\delta x}_{j_2} + (1 - \xi - \eta)\vec{\delta x}_{j_3} \tag{2.17}$$

with the addition to the system stiffness matrix as previously shown.

2.3 Suitable Spring Stiffnesses

The final component of the system stiffness matrix is the formulation of the spring stiffness. A desirable property of the system is that edges and elements should not collapse. Thus, a dynamic stiffness is highly desirable, where an increase in resistance is sought as edges and/or elements approach collapse and decrease in the reverse. Making use of the inverse of the edge length does exactly this. Thus the formulation for the a spring stiffness is

$$k_e = \frac{1}{L_e^n} \tag{2.18}$$

and is used for both edge and element springs. It is noted that the above is non-linear and therefore inherits the complexity therein. While the above formulation has been previously noted [37,38], typical selection of n is 1 [1,34–36,39,40,42]. For use here, it was found that a value of 2 produced better results. Thus, stiffnesses for the presented scheme are formulated as

$$k_e = \frac{1}{L_e^2} \tag{2.19}$$

This change creates a greater difference in relative stiffness, which was found to be imperative for the industrial CFD test cases. The majority of the domain is of little interest and edges in these uninteresting areas should give more. Conversely, forces resulting from sonic shocks are very large, and the relatively smaller elements should thus avoid crushing.

2.4 Monitor Function: Error Estimator

2.4.1 General Force Formulation

For any mesh optimisation, some metric or monitor function must be used to drive the nodes to the desired areas of the mesh. In this work, the latter (which is indicative of error) is represented as a force applied to nodes as per Acikgoz [42]. With reference to CFD, Zienkiewicz and Zhu [5] were the first to note that the flow gradients are useful as error estimates. However, Roy [44] has shown that while gradients are indicative of error, the result on adaptivity could be a deterioration in accuracy. Roy demonstrates that this may be corrected by employing an expression which is more representative of the actual discretization or truncated error. In the case of smooth fields, second derivative or curvature was found to also be of great importance.

For the purposes of this work, a balance was sought between complexity of error estimator and accuracy. The proposed formulation therefore hinged on the observation that curvature is indicative of error for smooth fields, while gradient is needed to cluster nodes around discontinuities (sonic shocks in the case of compressible flow.) For fields that contain both, therefore, the following intuitive hybrid expression for error was formulated across edges:

$$\vec{f}_{ij} = C_c \vec{f}_{ij,curve} + C_g \vec{f}_{ij,grad} \tag{2.20}$$

where $\vec{f}_{ij,curve}$ and $\vec{f}_{ij,grad}$ represent the forces over the edge due to the curvature and gradient of the flow field over the edge respectively. C_c and C_g are the curvature and gradient adjustment coefficients and are discussed shortly in Section 2.4.4. The resulting forces can be seen overlaid on an arbitrary node in Figure 2.5.

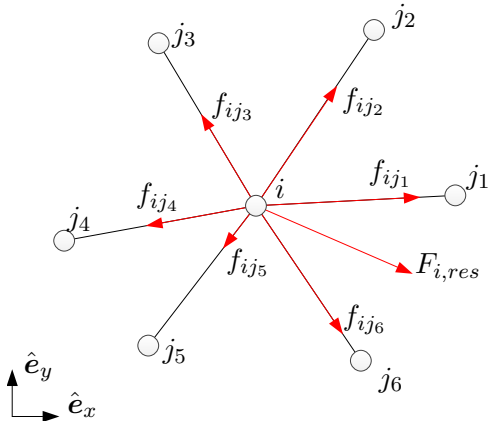

Fig. 2.5: Typical loading of a node with only edge springs being used.

$$\vec{F}_{i,res} = \sum_{j_0}^{j_m} \vec{f}_{ij_e} \tag{2.21}$$

It should be noted that the edge force f_{ij_e} should always be compressive and act to shorten the spring. The logic here: a smaller edge/spring produces a smaller error, since the error is a function of mesh spacing [5].

For this work transonic compressible flow (containing transonic shocks) was considered. Thus, the flow density was employed as the monitored variable. Clearly, for different CFD applications, this is adaptable so long as the functional is reducible to a vector of the above form. To maintain generality, ϕ is used to indicate the monitored flow variable, typically at a node.

2.4.2 Gradient monitor function

The reconstruction of gradient over an edge can be done simply by using difference in flow property over the edge.

$$\vec{f}_{ij,grad} = -\vec{f}_{ji,grad} = \frac{|\phi_i - \phi_j|}{|\vec{x}_i - \vec{x}_j|}\vec{t}_{ij} \tag{2.22}$$

and for a monitor function, where ϕ is some vector property, the force can be expressed as

$$\vec{f}_{ij,grad} = \frac{\sqrt{\sum_d \left((\vec{\phi}_i - \vec{\phi}_j) \cdot \hat{\mathbf{e}}_d\right)^2}}{|\vec{x}_i - \vec{x}_j|}\vec{t}_{ij} \tag{2.23}$$

Again, ϕ in the above system can represent any flow property on which to base the monitor function.

2.4.3 Curvature Monitor Function

The reconstruction of the curvature is more challenging compared to gradient. Although Roy [44] employed second derivative for 1D, multi dimensional applications require curvature. Mathematically, this is defined for some polynomial $\phi = g(l)$ as

$$\varkappa = \frac{\phi''}{(1+\phi'^2)^{\frac{3}{2}}} \approx \frac{d^2y}{dl^2} \tag{2.24}$$

where ϕ' and ϕ'' are the first and second derivatives of the curve $g(l)$ and l denotes edge length. The curvature $\varkappa$ is calculated by reconstructing the function $g(l)$ across the edge. Here, it is desirable that the field maintains smoothness across the various attached elements. This is employed by using a clamped cubic spline:

$$g(l) = a_0 l^3 + a_1 l^2 + a_2 l + a_3 \tag{2.25}$$

where the above coefficients a_0 to a_3 are the polynomial coefficients to be solved. To construct a clamped spline, directional derivatives at the ends are required and are of the form

$$\nabla_{ij} g(l) = \nabla g(l) \cdot \vec{t}_{ij} \tag{2.26}$$

with l varying between 0 and L_{ij}; the polynomial coefficients may be computed as

$$\begin{aligned}
a_0 &= \frac{2}{L_{ij}^3}(\phi_i - \phi_j) + \frac{1}{L_{ip}^2}\nabla_{ij}\left[g(0) + g(L_{ij})\right] \\
a_1 &= \frac{3}{L_{ij}^2}(\phi_j - \phi_i) - \frac{1}{L_{ij}}\nabla_{ij}\left[2g(0) + g(L_{ij})\right] \\
a_2 &= \nabla_{ij} g(0) \\
a_3 &= \phi_i
\end{aligned} \tag{2.27}$$

where ϕ_i and ϕ_j denote field variable values at the nodes. The derivatives of g may then be calculated as

$$\begin{aligned}
g'(l) &= 3a_0 l^2 + 2a_1 l + a_2 \\
g''(l) &= 6a_0 l + 2
\end{aligned} \tag{2.28}$$

In the interest of consistency, the edge spring force resulting from curvature is computed at $l = 0.5L_{ij}$ as

$$\vec{f}_{ij,curve} = -\vec{f}_{ji,curve} = \varkappa|_{0.5L}\vec{t}_{ij} \tag{2.29}$$

with nomenclature previously defined.

2.4.4 Adjustment Coefficients

Due to the non-physical nature of both forces and truss stiffnesses, it is to be ensured that these are kept in balance with respect to each other. To achieve this, an adjustment factor C_{adj} (see Section 2.5.1) must be used to bring the RHS and the LHS of Equation (2.5) in line. Additionally, the adjustment factor is used on the first iteration to set the adjustment coefficients C_g and C_c. Critical importance is placed on the ratio between C_g and C_c (with the absolute value of each therefore being of lesser value), as will be investigated in Chapter 5.

2.5 Moving Meshes

2.5.1 Preventing Element Collapse

The edge and element springs clearly displace in a non-linear fashion (due to non-linear varying stiffness), but linear solvers do not support this. Therefore, nodes are restricted only to move some factor of local element spacing. For the purposes of this work, the latter is defined as the safety factor η_{sf}. The displacement of a node is limited by taking the 'length to collapse' and dividing it by the safety factor. The 'length to collapse' L_{cl} is defined as the minimum distance a node would need to travel to cause any of its surrounding elements to become concave, cross an edge in any of its surrounding elements, or create an edge of zero length, as shown below in Figure 2.6.

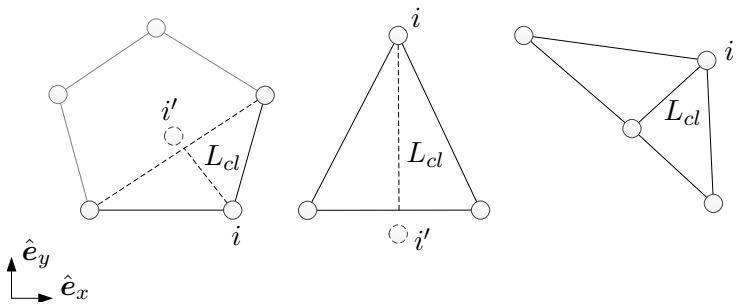

Fig. 2.6: The movement of node i to i' causing a concave element (left), a node-edge crossing (center), and node-node collision (right).

L_{cl} is simply L_{ip} from Equation (2.9) for all element configurations. Thus, no further calculations are required to obtain L_{cl} and the maximum allowable nodal displacement $\vec{\delta x}_{i,m}$ is given by

$$\vec{\delta x}_{i,m} = \frac{min(L_{ip})}{\eta_{sf}} \tag{2.30}$$

Once the nodal displacement solution $\vec{\delta x}$ has been obtained, it is checked against the node's maximum allowable distance. Should it be greater, it is adjusted to be equal to $\vec{\delta x}_{i,m}$ as shown by Figure 2.7.

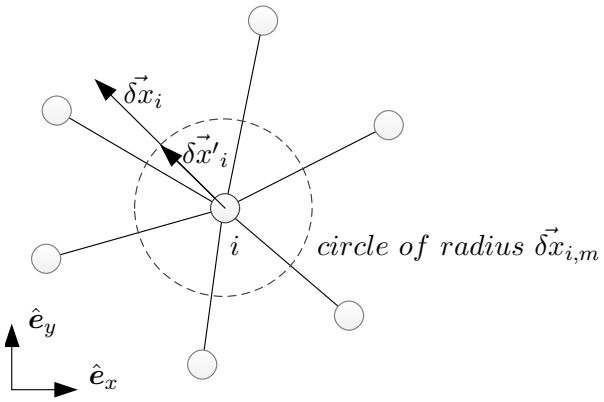

Fig. 2.7: Node displacement $\vec{\delta x}_i$, correction for a solution bigger than the maximum allowable displacement $\vec{\delta x'}_i$.

To effect, the above $\vec{\delta x}_i$ is multiplied by a constant $C_{i,adj}$, the adjustment factor. To maintain relative motion between nodes, a single constant C_{adj} must be used through the entire mesh. Clearly, the solution vector, which

used the smallest $C_{i,adj}$, must be selected in order to maintain mesh validity. The new node positions can then be determined using

$$\begin{aligned} \vec{x'}_i &= \vec{x}_i + C_{adj}\vec{\delta x}_i \\ C_{adj} &= min\left(C_{i,adj}\right) \end{aligned} \tag{2.31}$$

On the first iteration, the node with the smallest $C_{i,adj}$ is moved to its circle of maximum radius (see Figure 2.7) and C_{adj} set accordingly for the entire mesh.

2.5.2 Laplacian Smoothing

Although the ball-vertex method does, even under severe deformation, generally produce a valid mesh, it can in some cases reduce element quality to a point where the flow solver becomes unable to re-converge the solution. In an attempt to alleviate this problem, a laplacian smoother was applied to the mesh:

$$\vec{x'}_i = \frac{1}{E}\sum_j \vec{x}_j \tag{2.32}$$

The above, however, can occasionally produce invalid elements. Liakioiulos and Giannakoglou [45] used a reduced polygon to constrain the smoothing. The points selected for the smoothing are the vertices of the reduced polygon which are shown in Figure 2.8.

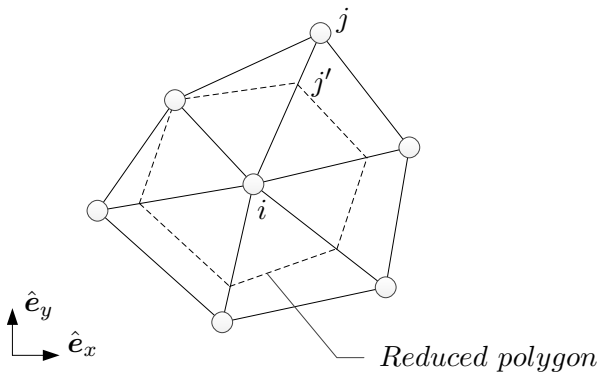

Fig. 2.8: Showing the modified polygon from which smoothing points are selected.

The new coordinates are accordingly updated as

$$\begin{aligned} L_{ij,min} &= min(|\vec{x}_i - \vec{x}_j|) \\ \vec{x'}_j &= \vec{x}_i + L_{ij,min}\vec{t}_{ij} \end{aligned} \tag{2.33}$$

and $\vec{x'}_j$ is substituted directly back into into Equation (2.32), as $\vec{x}_j$.

3 Boundary Node Refinement

3.1 Introduction

The ability to refine boundaries is an essential part of an r-refinement scheme. In a typical CFD simulation, the functions that describe the boundary geometry are not normally known, yet the nodes must move in such a way to ensure that fidelity of the geometric surface is maintained. Further, since the boundary is updated regularly during an r-refinement cycle, the method used to calculate geometric features must be automated and must be robust. Hence, this chapter first deals with some of the new geometric identification techniques which were implemented before continuing on to detail the methodology for updating boundary nodes in the proposed r-refinement scheme.

3.2 Boundary Geometric Identification

The primary purpose of feature identification is to define corners and ridges and create two sets of boundary normals. The first set composes outward pointing unit normals at every boundary node which are named the boundary node normals, denoted $\vec{N}$. The second set consists of face normals for every boundary node on each boundary face (which are stored at face nodes as per Figure 3.1), denoted $\vec{n}$ and named boundary face normals. Additional information pertaining to the definitions of boundary faces and to element numbering format used is presented in Appendix A.

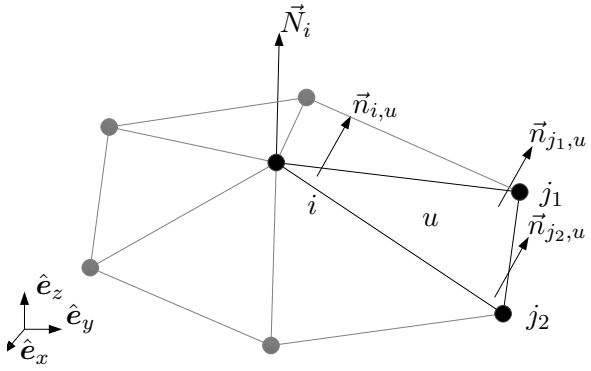

Fig. 3.1: Boundary and face normals associated with boundary face u. Greyed nodes/edges serve to offer background context.

For clarity of illustration, mesh entities (nodes and edges) which serve only to give background context to a figure are coloured in grey. Further, boundary nodes are shaded in black or grey while interior nodes are not (as per Figure 3.4).

Boundary face normals $\vec{n}_{i,u}$ are constructed for face u using

$$\vec{n}_{i,u} = \frac{\vec{t_{ij_2}} \times \vec{t_{ij_1}}}{\left|\vec{t_{ij_2}} \times \vec{t_{ij_1}}\right|} \tag{3.1}$$

with $\vec{t}$ defining the edge vectors from node i to the connected nodes j_1 and j_2.

3.2.1 Feature Properties

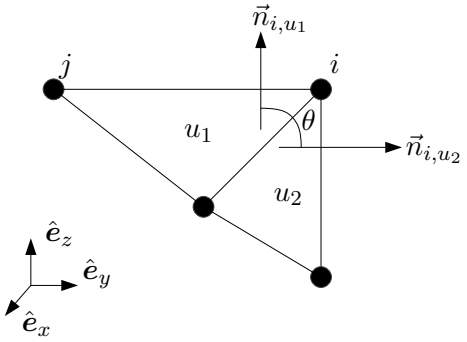

Fig. 3.2: Ridge detection test using two boundary face normals.

With the boundary face normals, $\vec{n}_{i,u_1}$ and $\vec{n}_{i,u_2}$, the angle θ is constructed between two boundary faces, which share a common edge. Thus, θ is given

by

$$\theta = \arccos\left(\frac{\vec{n}_{i,u_1} \cdot \vec{n}_{i,u_2}}{|\vec{n}_{i,u_1}|\,|\vec{n}_{i,u_2}|}\right) \tag{3.2}$$

The boundary node normal is next calculated, though the exact formulation is dependent on the boundary node type: smooth, ridge or corner. In the case of smooth boundaries, the node normal is computed as

$$\vec{N_i} = \sum_u \frac{|\vec{t}_{ij_1,u} \times \vec{t}_{ij_2,u}| \vec{n_{i,u}}}{\sum_u |\vec{t}_{ij_1,u} \times \vec{t}_{ij_2,u}|} \tag{3.3}$$

where u denotes the attached boundary faces.

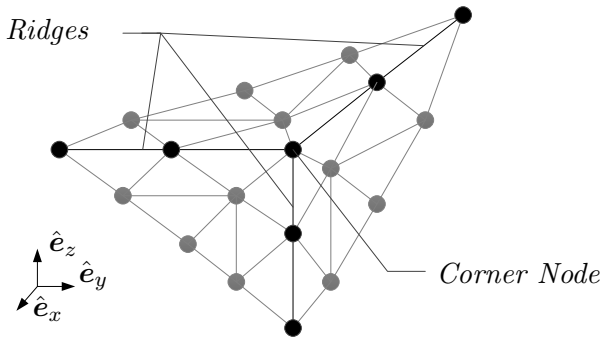

Fig. 3.3: A corner node surrounded by three ridges.

Corners and ridges (non-smooth) occur when adjacent boundary face normal angles, θ, differ by more than a user specified threshold. An example of the boundary faces of a tetrahedral mesh are shown in Figure 3.3.

Ridge nodes (ridges) occur when the attached boundary faces may be separated into two so called "face groups". A face group, denoted λ, constitutes a set of boundary faces which are attached to a boundary node and have face normals which are equal or differ by a small amount (less than the threshold angle). For each group, an area weighted average vector is constructed (using Equation (3.3)). The two face group normals are then added together and renormalized to obtain the outward pointing boundary node normal $\vec{N_i}$, given by

$$\vec{N_i} = 0.5 \sum_{\lambda=1}^{2} \vec{n}_{i,\lambda} \tag{3.4}$$

Corners occur when more than two face groups are attached to a boundary node. The outward pointing node normals are similarly calculated. Finally, boundary face normals (which are stored at boundary face level at each node) are set equal to the face group value.

3.3 A Priori Algebraic Constraints

The proposed scheme manipulates various solution steps in both an *a priori* and *a posteriori* manner, with respect to iterations, increasing the responsiveness of boundary node refinement. *A priori* constraints are applied at discretization level (prior to solution) and serve to ensure that a node is limited to slide along a specific flat plain or remain fixed in position at a corner. Slight departures from the geometry may still occur, particularly in areas of high boundary curvature. Here *a posteriori* treatment is applied to bring the node back onto the boundary geometry, enforcing strict boundary adherence even in cases of curved surfaces.

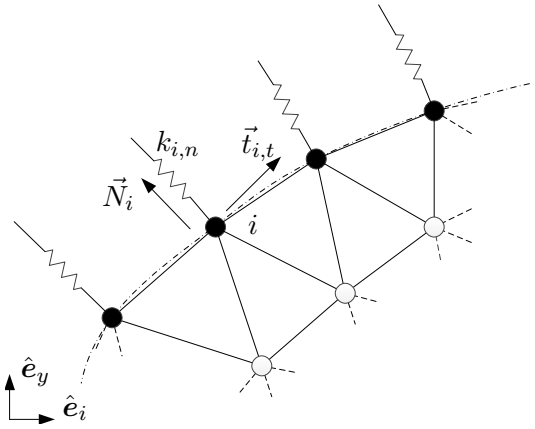

Fig. 3.4: Boundary nodes with attached boundary springs of stiffness k

To enact the above, both the system stiffness matrix and load vector must be suitably adjusted. The resulting error force (Equation (2.21)) at each boundary node is modified by the removal of its boundary normal component. The resultant applied force (due the monitor function) will therefore not push or pull the node off the surface. Using the boundary node normal $\vec{N}_i$ the resultant force $\vec{F}_i$ at the boundary node i is modified such that

$$\vec{F'}_i = \vec{F}_i - \vec{F}_i \cdot \vec{N}_i \tag{3.5}$$

Normal movement can still occur due to internal mesh movement. *A priori* constraints are achieved by attaching a set of springs which are in the direction of the boundary node normal, $\vec{N}_i$. These springs are then given a stiffness value which is three orders of magnitude larger than any attached spring and inserted into the system stiffness matrix diagonal. In 2D this reads:

$$\bar{\boldsymbol{K}}'_i = \bar{\boldsymbol{K}}_i + k_{i,n} \begin{pmatrix} \vec{N}_{i,x} \cdot \vec{N}_{i,x} & \vec{N}_{i,x} \cdot \vec{N}_{i,y} \\ \vec{N}_{i,y} \cdot \vec{N}_{i,x} & \vec{N}_{i,y} \cdot \vec{N}_{i,y} \end{pmatrix} \tag{3.6}$$

It is to be noted that the above is applied only at smooth boundary nodes. In the case of corner nodes, all displacements are constrained via a Dirichlet condition ($\vec{\Delta x} = 0$). Algebraically, this is achieved by setting all off-diagonal entries in the system stiffness matrix as well as the load vector to zero. Conversely, ridge node displacement can be allowed so long as the geometric ridge is preserved. To achieve this, two springs in the direction of the two face group normals are added, as shown in Figure 3.5.

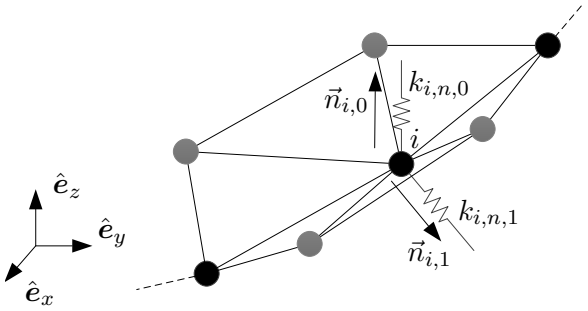

Fig. 3.5: A ridge node constrained by two attached ridge springs with stiffness k, each aligned parallel to the face group normals n.

3.4 A Posteriori Geometric Constraints

As noted previously, *a posteriori* treatment is required in certain instances to ensure that the boundary nodes remain strictly on the geometric boundary. The overarching process can be split into three separate steps. Firstly, the boundary face over which the node has moved (displacement direction) is identified, secondly the node's geometric classification (smooth, ridge or corner) and thus its displacement type is determined. Lastly, the boundary node is placed back onto the boundary surface where required.

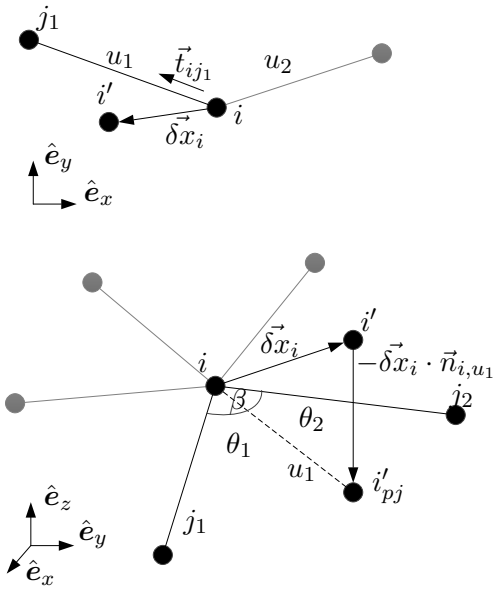

Fig. 3.6: Node i displaced off the boundary surface to an arbitrary point i' in 2D (top) and 3D (bottom).

To begin with, the boundary face over which a node has moved (Figure 3.6) is identified. To do this, a set of unit edge vectors $\vec{t}_{ij}$ are constructed from node i to surrounding boundary nodes. The boundary face over which node i has moved will yield the largest value of a dot product between $\vec{t}_{ij}$ and $\delta\vec{x}_i$. This test is referred to as the dot-product test further on. In 3D (Figure 3.6), the problem is more complex. All the boundary elements attached to node i are visited. In each element, the ideal placement of node i, i' is projected onto the boundary element face using the boundary face normals at node i by

$$\vec{x'}_{i_{pj}} = \vec{x'}_i - \delta\vec{x}_i \cdot \vec{n}_{i,u_1} \tag{3.7}$$

The sought-after face is that for which the following relation is satisfied:

$$\theta_1 + \theta_2 = \beta \tag{3.8}$$

The above is referred to as the angle test.

Having identified the boundary face over which a node has moved, consideration is given to boundary classification: corner, ridge or smooth. However, due to the computational cost associated with a curved smooth surface (as opposed to a flat plane), additional classifications are employed. These are described next and with reference to Figure 3.7. For this reason, linear displacement is handled separately and therefore needs to be detected.

Noted is that even with the *a priori* approach, it is still expected that there will be normal motion, which must be considered.

Nodes on a smooth boundary are those nodes which are not corner or ridge nodes. These nodes often make up the majority of all boundary nodes.

Nodes on a straight line/flat plane (2D/3D) are special cases of smooth boundary nodes and occur when the boundary node normals of surrounding nodes are found to be parallel.

Nodes on straight ridges are nodes, tagged as ridges, where the connected ridge nodes have node normals which are parallel.

Nodes on smooth ridges are all other ridge nodes.

Nodes on corners are not moved.

Nodes with an undesirable displacement solution are nodes where the dot-product test (2D) or the angle test (3D) fail to provide a boundary face over which the boundary node has moved. In 2D, this is identified when the dot-product of all connected boundary nodes are negative; and in 3D, when none of the surrounding boundary elements satisfy Equation (3.8).

Finally, the actual process of updating the boundary can be done. The six movement cases, identified previously, are used to create a placement procedure for every boundary node. In the typical case, the position to which a node is to be moved is taken as the closest position on the geometric surface. In the case of general (curved) smoothed surfaces, third order Bèzier curve or surface fits are used in conjunction with a minimisation procedure, described shortly. For linear boundaries, nodes are placed at their boundary projected locations. Finally, nodes on corners as well as undesirables are not moved. A more detailed break down of each case's specific placement procedure can be found in Appendix B.

Returning to curved (general) smooth boundaries; as noted, third order Bèzier curves are employed. To find the closest position on the surface, a minimisation procedure using Newton linearisation, is carried out. For further information regarding Bèzier control points, refer to Appendix C.

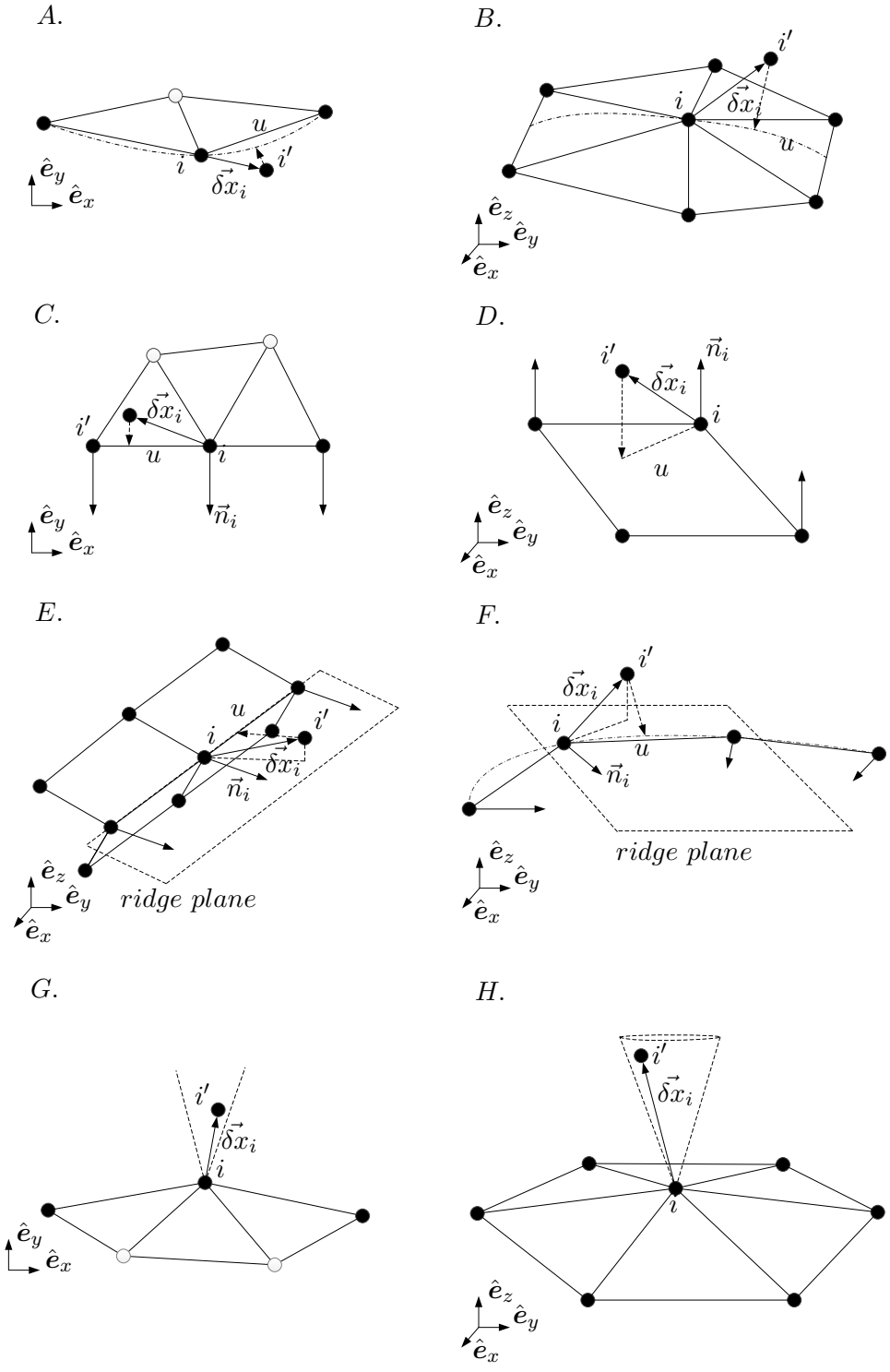

Fig. 3.7: Various displacement solutions ($\vec{\delta x}_i$) of node i over boundary element u and the subsequent placement back onto the boundary surface shown for, *A*. Smooth 2D, *B*. Smooth 3D, *C*. Straight line, *D*. Flat plane, *E*. Straight ridge, *F*. Smooth ridge, *G*. Undesirable 2D, *H*. Undesirable 3D

4 Solution Procedure

The developed r-refinement scheme is next integrated into the CFD flow solver, Elemental^{TM} (see Figure 4.1). Once the flow solver has been initialised, feature identification takes place. The flow solution is converged in a typical manner until it meets a user specified threshold, at which point the r-refinement scheme is called. At present, the refinement scheme is called a set number of times. Once the last call has been made, the flow solution is converged fully.

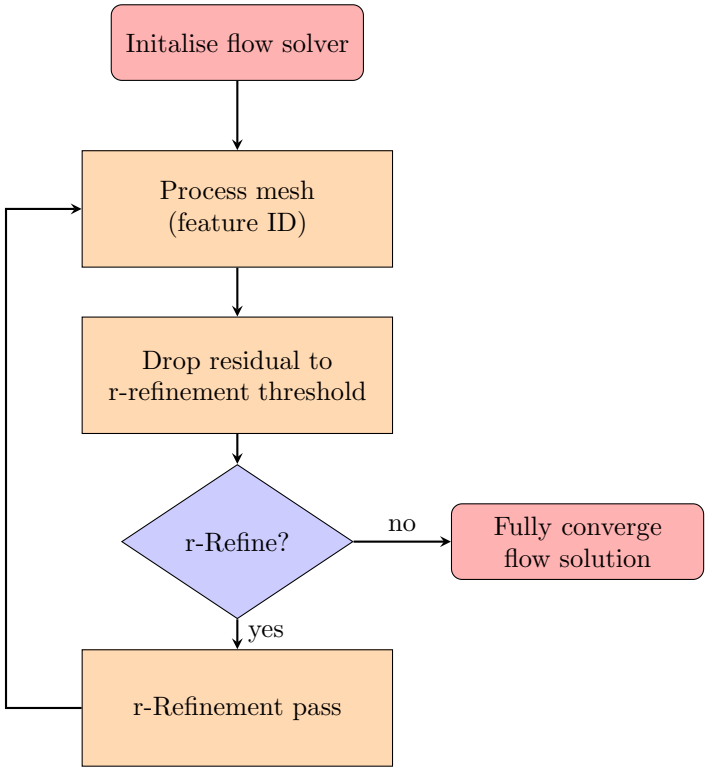

Fig. 4.1: Refinement scheme integration with the flow solver; the user to specify the number of r-refinement passes required.

The refinement pass in Figure 4.1 may be sub-divided into three main sub-functions (see Figure 4.2): building and solving the truss network, ensuring nodes move correctly and the *a posteriori* boundary node treatment. Other processes are also shown, such as the updating of geometry which, in turn, involves re-computation of the edge lengths and nodal volumes. The laplacian smoothing is employed once r-refinement convergence has been reached. Also shown is the interpolation of the field variables, discussed in Section 4.4.

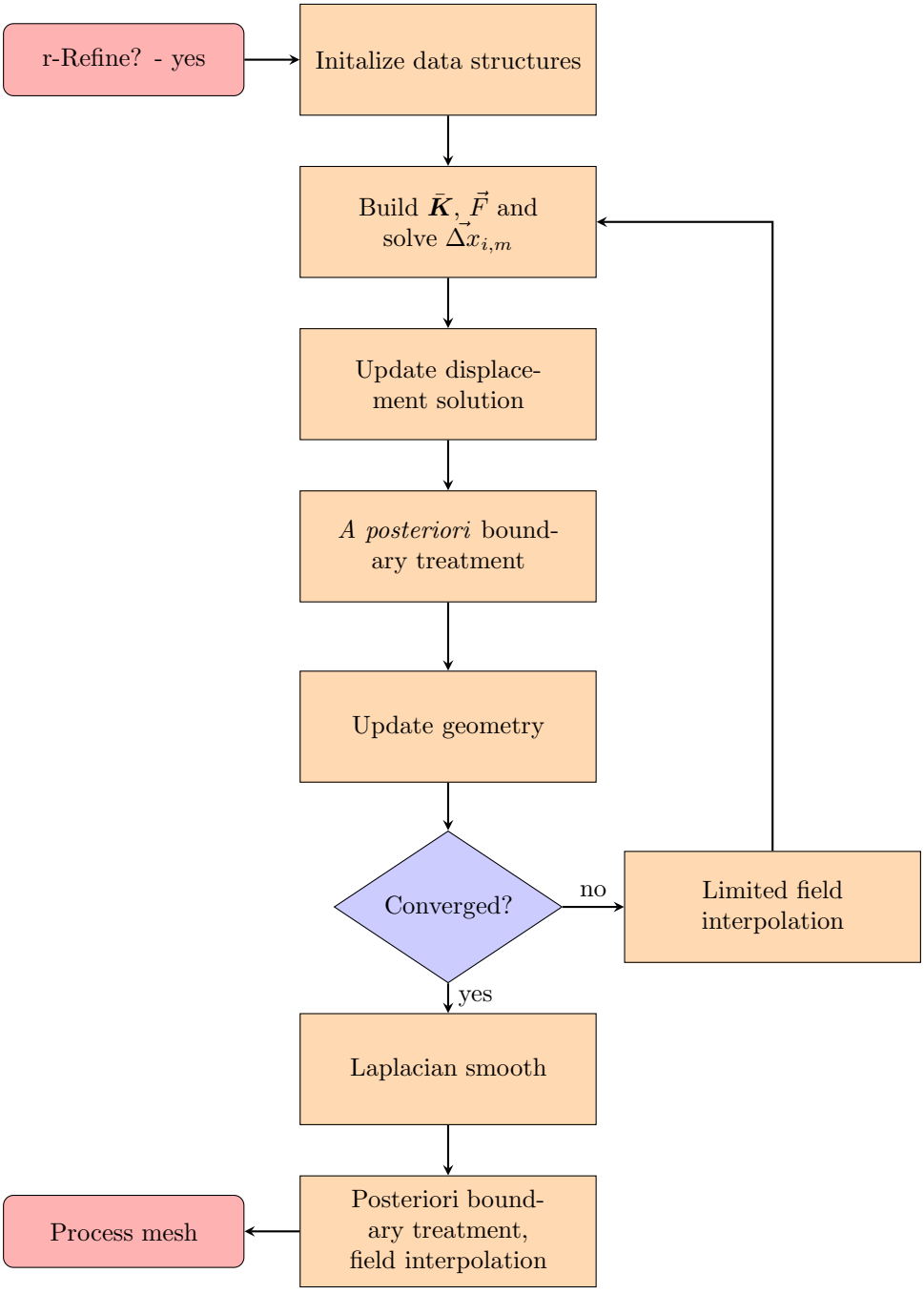

Fig. 4.2: The refinement pass functional flow.

4.1 Solution Methodology

Recalling from Section 2.1, a general solution procedure would be of the form

$$\bar{\boldsymbol{K}}^{i+1}\vec{\Delta x}^{\,i+1} = \vec{F}^{i+1} - \bar{\boldsymbol{K}}^{i}\vec{\Delta x}^{\,i} \tag{4.1}$$

where

$$\vec{\Delta x}^{\,i} = \vec{x}^{i} - \vec{x}^{0}$$

Here, $\vec{\Delta x}^{\,i+1}$ should, over a few iterations, converge to zero. Note that the above iterative procedure (and re-calculation of $\bar{\boldsymbol{K}}^{i}$) is required due to the non-linear nature of the problem (materially and geometrically). However, one may recall the assumption made with regard to the absence of rotation in a spring member's stiffness. Since large deflections are expected, this assumption needs to be re-examined. For the purpose of this work, it is proposed to alleviate this by mixing the original system stiffness matrix with that of the new (by implication the stiffness magnitude would also be a mixture). An equal weighting is employed as it would be accurate for pure rotations as:

$$\bar{\boldsymbol{K}}^{i} = \frac{1}{2}\left(\bar{\boldsymbol{K}}^{i+1} + \bar{\boldsymbol{K}}^{0}\right) \tag{4.2}$$

where $\bar{\boldsymbol{K}}^{0}$ represents the system stiffness matrix on the first iteration. For solution purposes, the Elemental$^{\text{TM}}$ preconditioned GMRES as well as AMG solvers were employed. In the above, $\bar{\boldsymbol{K}}^{0}$ may be sampled at the first iteration of the first r-refinement pass, or at the first iteration of every r-refinement pass (see Figure 4.3). Both of these are evaluated in the results chapter to assess efficacy.

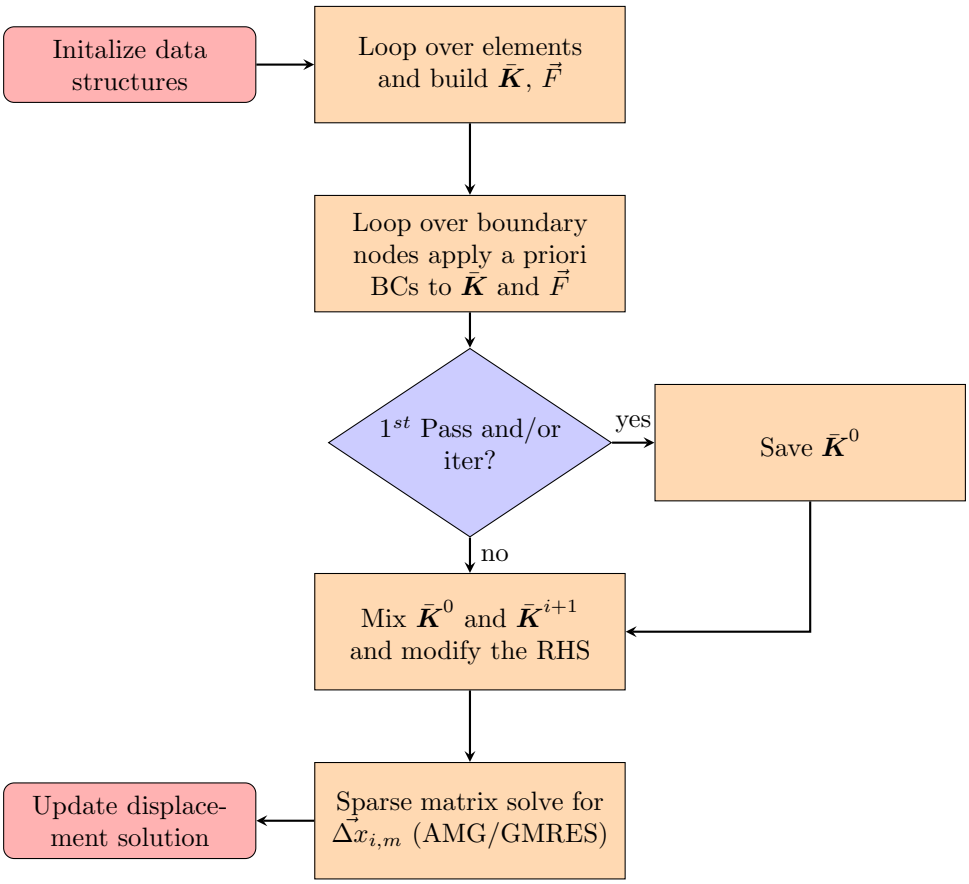

Fig. 4.3: The building of $\bar{\boldsymbol{K}}$, $\vec{F}$ and solving for $\vec{\Delta x}_{i,m}$ functional flow.

4.2 Solution Processing

Figure 4.4 shows the algorithm for ensuring that nodes stay within their ball as discussed in Section 2.5. Additionally, the setting of C_g and C_c takes place on the first iteration as shown. Finally, the reader is reminded that the adjustment factor C_{adj} is applied to all nodes in the mesh.

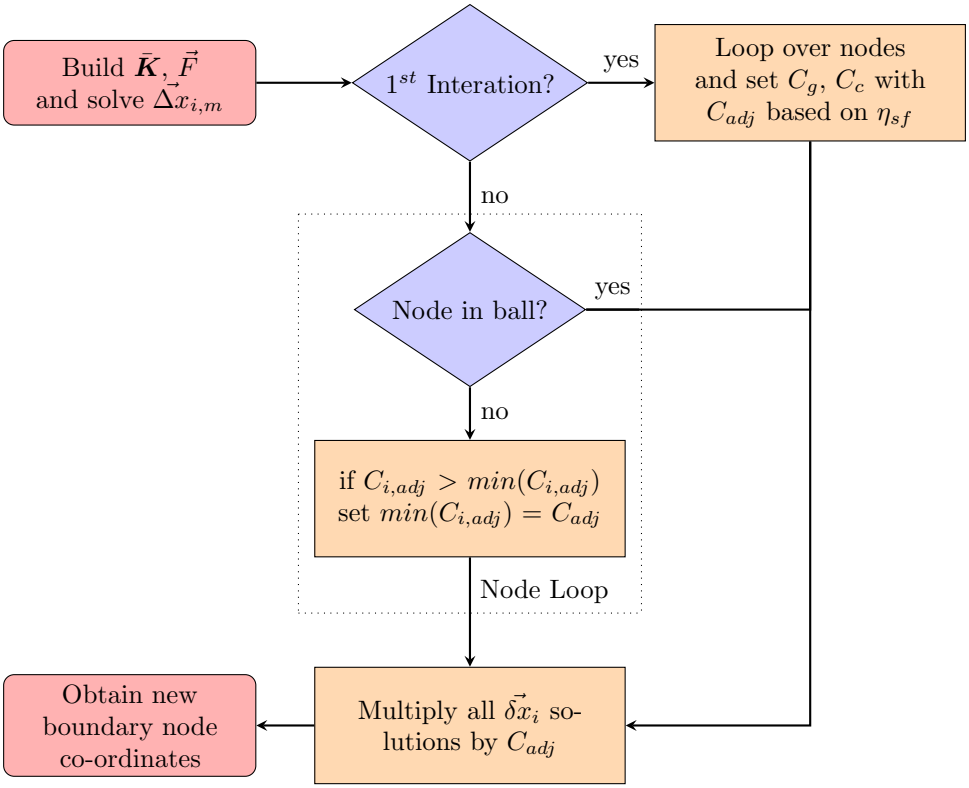

Fig. 4.4: The process for limiting nodal displacement

4.3 Boundary Node Placement

The last major algorithm (see Figure 4.5) is that which places nodes back onto the boundary as described in Chapter 3, which forms part of the general co-ordinate updating procedure. First, the node classification is sought before the boundary element over which the node has moved is determined. After which the new node is placed back onto the boundary.

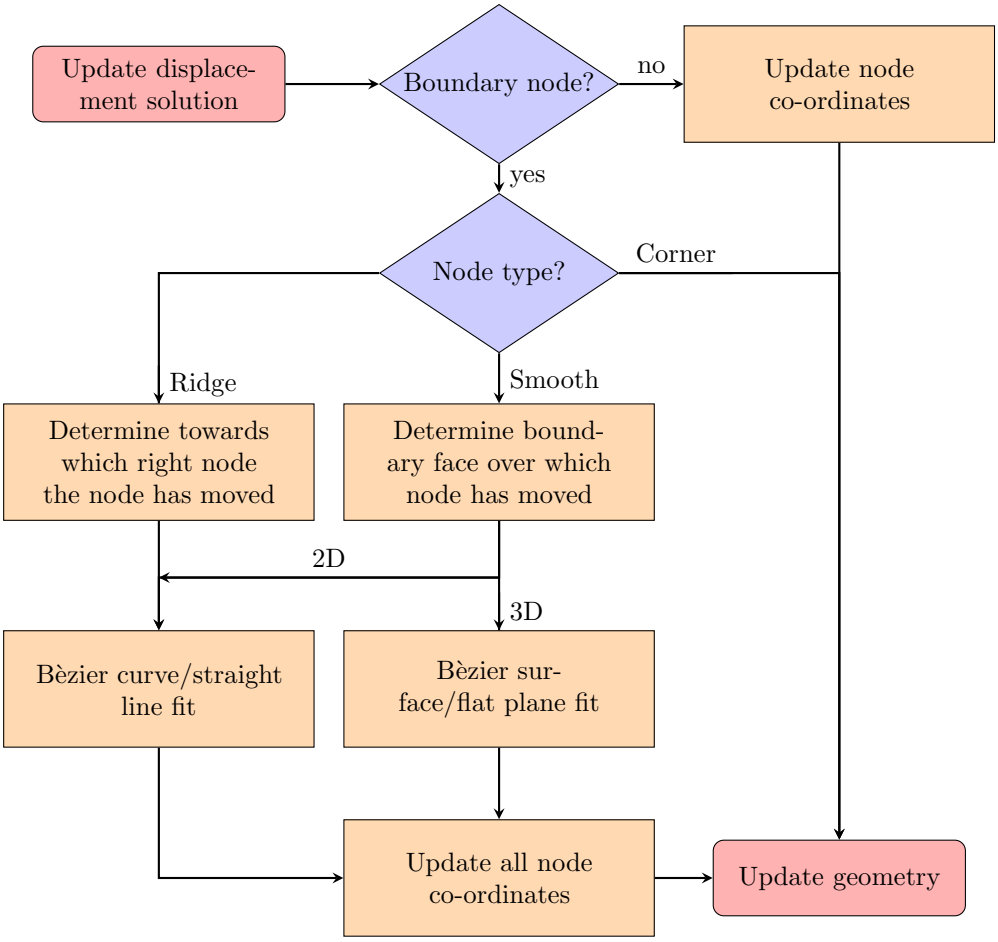

Fig. 4.5: Work flow for finding new boundary node co-ordinates when looping over all nodes.

4.4 Field Interpolation

To avoid the additional computational overhead of re-computing field variables (used for the monitor function) between successive r-refinement iterations, limited interpolation is employed - in the sense that only the field variable employed for the monitor function is interpolated. When handing back the refined mesh, however, all flow variables are interpolated forward.

Inverse distance interpolation was employed by using the original field variables (as received from the flow solver). This is such that repeated interpolation does not deteriorate the accuracy of the monitor function.

The interpolation equation employed is then

$$\phi_i^{i+1} = \frac{\sum_j \phi_j^0 |\vec{x}_i^{i+1} - \vec{x}_j^0|^2}{\sum_j |\vec{x}_i^{i+1} - \vec{x}_j^0|^2}$$
$$\sum_j |\vec{x}_i^{i+1} - \vec{x}_j^0|^2 \neq 0 \tag{4.3}$$

where subscript j and superscript 0 represent node data from the original mesh which surrounds the node i, at its position for the current iteration, superscripted $i+1$.

5 Numerical Examples

5.1 Overview

The developed r-refinement scheme is now evaluated via application to a number of test cases. To begin with, 2D analytical cases are considered in order to qualitatively demonstrate scheme capability. These tests are less about accuracy and more a demonstration of the ability to effect large node motion. r-Refinement is then applied to the NACA0012 aerofoil under transonic flow conditions, which constitutes a demanding, industrially relevant, 2D problem. Next, a 3D transonic flow case is considered, viz. flow over an aerofoil from the recently completed FP7 project: Future Fast Aeroelstic Simulation Technologies (FFAST) [46]. In the interest of a rigorous evaluation, improvements in accuracy (as compared to the mesh independent solution) are quantified as a function of a number of r-refinement passes as well as gradient to curvature ratio. This is done with coarse and fine meshes as starting points. Finally, a demonstration of the scheme as a mesh deformation tool is presented by means of an unrealistically large deflection of the 3D aerofoil.

5.2 2D Analytical Test Cases

As explained, the main focus of the analytical examples was qualitative and more on capability rather than exact reduction in error. The main objective was to demonstrate that, given some field, the r-refinement scheme can track and adapt accordingly. Additionally, the tests were to demonstrate that even in the presence of shock-like structures which feature high gradients, the mesh remains valid.

The first test case featured a smooth sinusoidal field and was a simple initial demonstration of viability. The second test case featured a field with a large field magnitude range and shock-like topology, which was taken from the r-refinement paper by G.L. Delzanno et al [30]. Both analytical scalar fields were examined in a unit 2D domain. Since density was the monitor function base for the aerofoil problems which follow, it was used here as the analytical field. For the purpose of the test case, error topology (monitor

function) was assumed to be based on gradient.

5.2.1 Sinusoidal Case

The scalar field $\rho(x, y)$ is given values corresponding to a cosine function oriented along the line $y = x$, expressed as

$$\rho(x,y) = 1.5 - 0.5\cos\left(4\pi\sqrt{\left(x - \frac{1}{2}(x+y)\right)^2 + \left(y - \frac{1}{2}(x+y)\right)^2}\right) \quad (5.1)$$

The density field and resulting fabricated error field are presented below (Figure 5.1). The field is overlaid on an equispaced (finite difference) mesh, with the resulting five and ten pass r-refined meshes shown in Figure 5.2.

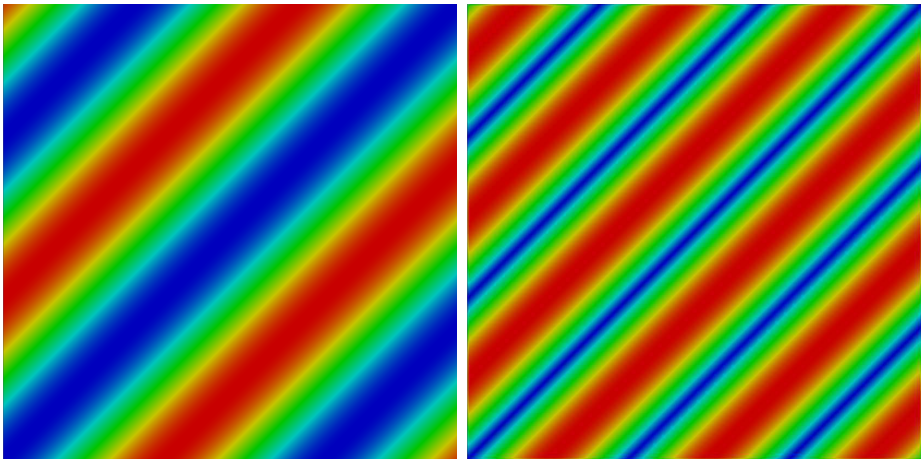

Fig. 5.1: Plot of the density field (left) and the error field (right) with red and blue indicating higher and lower values respectively in both plots.

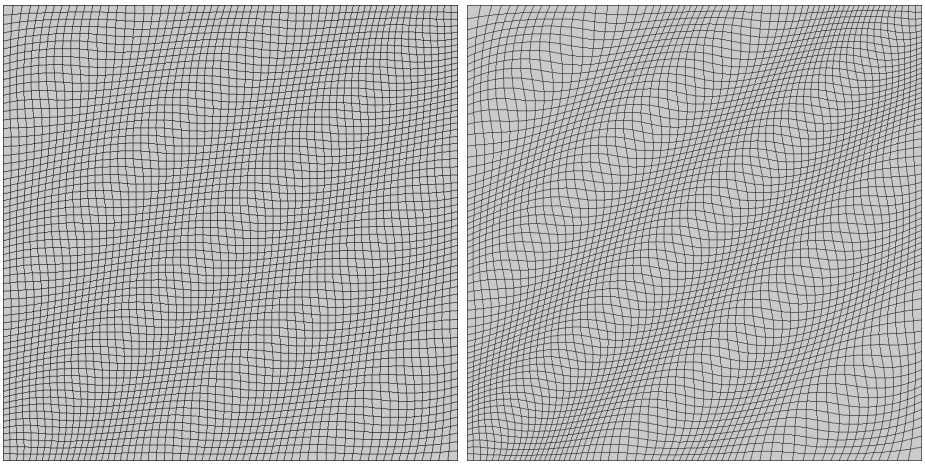

Fig. 5.2: Adapted mesh after 5 passes (left) and 10 passes (right) of r-refinement.

As can be seen from the adapted meshes, the resulting node distribution corresponds well with the error topology plot. In addition, as more passes were applied, the agreement in node distribution and error topology improved. Finally, commendable boundary motion was seen with boundary nodes adapting well and following the trend set by their non-boundary counterparts. Note that each pass required circa 5 iterations to converge, which required less than 2 seconds of CPU time on an Intel i7 2.30GHz chip with 8Gb of DDR3 RAM at 800MHz.

5.2.2 Vortex Case

As compressible flows can contain shocks which exhibit areas of high gradients and curvatures, a test case to simulate this was considered:

$$\rho(x,y) = \gamma\left(1 + \frac{9}{1 + [10r\cos(\theta - 20r^2)]}\right)$$

where

$$\begin{aligned} r(x,y) &= \sqrt{(x-0.5)^2 + (y-0.5)^2} \\ \theta(x,y) &= tan^{-1}\left(\frac{y-0.5}{x-0.5}\right) \\ \theta(0.5,y) &= \frac{1}{2}\pi \end{aligned} \tag{5.2}$$

and γ is the frequency factor which can be varied to change the wave length.

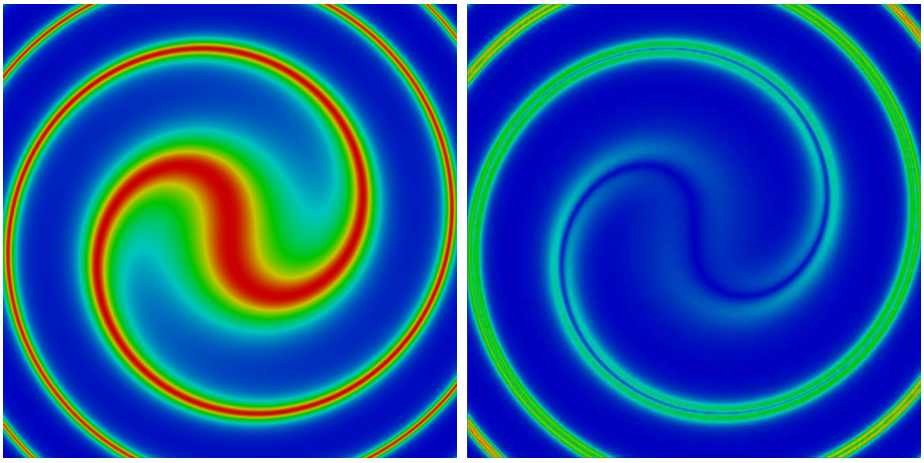

Fig. 5.3: Plot of the density field (left) and the error field (right) with red and blue indicating higher and lower values respectively in both plots

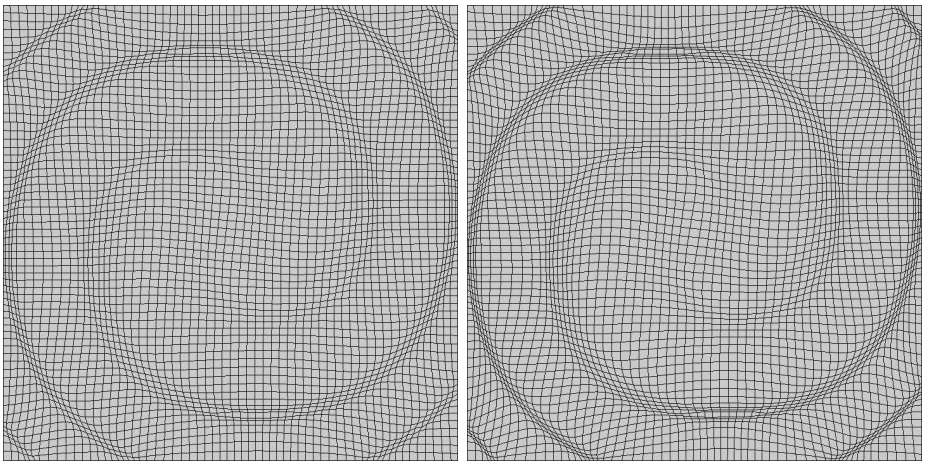

Fig. 5.4: Adapted mesh after 5 passes of r-refinement (left) and 10 passes (right).

Subsequent to r-refinement, the node distribution and error field again corresponded well. Additionally, as the error increased in the outer areas of the domain, so the node density increased.

5.3 2D NACA0012 Test Case

The industrial 2D example was that of the well known NACA0012 aerofoil. The flow was inviscid transonic, with the Mach number and angle of attack at 0.85 and 1.25° respectively. The 2^{nd} order accurate Elemental$^{\text{TM}}$ solver

was employed [47]. The success of the r-refinement algorithm was quantified via improvement in lift and moment. The target solution, i.e. "zero mesh spacing" solution, was obtained via the grid convergence index methodology [48] with Richardson's extrapolation.

Various aspects of the r-refinement scheme can be investigated. The most pertinent of these is the effect of varying the ratios C_c and C_g as discussed in Section 2.4.3. Additionally, the effectiveness of the Lapacian smoother on both accuracy as well as flow solver re-convergence was assessed together with the effect of resetting the system stiffness matrix. As mentioned, the monitor function was density-based.

5.3.1 Mesh Independent Solution

The mesh independence study involved the use of three Delaunay triangulation meshes with increasing node counts. A mesh of similar coarseness to the coarsest mesh is shown in Figure 5.6, with the density computed on the finest mesh depicted in Figure 5.5. The lift and moment coefficients were computed on the various meshes and are listed in Table 5.1.

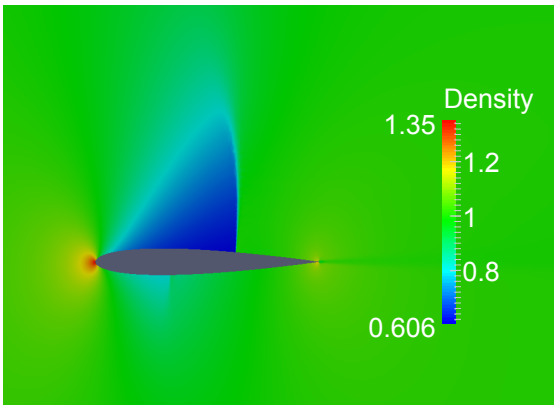

Fig. 5.5: Plot of the density field on a 72k node mesh for a NACA0012 aerofoil.

Tab. 5.1: Results from the various meshes for c_l and c_m in 2D.

No.	Nodes	c_l	c_m
1	5380	0.33734	-0.03748
2	30115	0.33935	-0.03757
3	72858	0.34234	-0.03784

With the above, the mesh independent solution may be calculated via Richardson extrapolation as per [48]:

$$\zeta_{extrap} \cong \zeta + \frac{\zeta_1 - \zeta_2}{r_{23}^p - 1} \tag{5.3}$$

Here, the variable for which the mesh independence is being sought is denoted ζ and is referred to as the the extrapolation variable. The grid refinement ratio r was calculated for the first two meshes and is thus given by

$$r_{12} = \left(\frac{i_{m,1}}{i_{m,2}}\right)^{\frac{1}{d_m}} \tag{5.4}$$

where i_m denotes the number of nodes in the mesh and d_m, again, the solution dimension. The refinement ratio of meshes 2 and 3, r_{23} was computed similarly. Additionally, the order of accuracy p in Equation (5.3) is expressed as

$$p^{j+1} = \omega p^j \left(1 - \omega\right) \frac{ln(\vartheta^j)}{ln(r)} \tag{5.5}$$

where

$$\vartheta^j = \frac{(r_{12}^{p^j} - 1)\epsilon_{23}}{(r_{23}^{p^j} - 1)\epsilon_{12}} \tag{5.6}$$

with j denoting the iteration number and ω the relaxation factor which is equal to 0.5. The initial guess for p^0 was 2 as this was the expected order of accuracy. For the above, ϵ represents the difference between the finer and coarser meshes' extrapolation variables, and is thus given by

$$\begin{aligned} \epsilon_{12} &= (\zeta_2 - \zeta_1) \\ \epsilon_{23} &= (\zeta_3 - \zeta_2) \end{aligned} \tag{5.7}$$

The certainty, or relative certainty, with which the extrapolated variable is judged can be calculated with the use of the grid convergence index or GCI as:

$$GCI = \eta_{sf} \frac{\left|\frac{\zeta_2 - \zeta_1}{\zeta_1}\right|}{r^p - 1} \tag{5.8}$$

where η_{sf} is selected as 1.25 as recommended by Roache [48].

The results obtained for the values c_l and c_m if using meshes 2 and 3 are given in Table 5.2.

Tab. 5.2: Results from the mesh independence study for c_l and c_m.

	c_l	c_m
ζ_{extrap}	0.3368	-0.0375
GCI	0.95%	0.34%

Having obtained the target solution, the developed r-refinement technology was applied to two meshes of different resolution. These are referred to as fine and coarse meshes respectively.

5.3.2 Fine Mesh Refinement

The mesh for the first refinement case was similar in coarseness to the coarsest mesh used in the mesh independence study. Here, the selection of 'fine' in the title is used to distinguish this mesh from the next case, where the mesh is significantly coarser. For these tests, a specified number of refinement passes were applied to a mesh. Each refinement pass occurred when the flow solution had re-converged to $1.0e-3$. Once all refinement passes were complete, the flow solution was converged to $1.0e-5$ to conclude the simulation. In a batch of tests, the number of refinement calls were kept constant.

The set-up for the refinement was therefore as follows:

1. Four batches consisting of two 5 and two 10 pass refinements are done.
2. One of the two sub-batches will have the Laplacian smoother activated and the other deactivated.
3. In each sub batch, 13 individual tests are done with varying $\frac{C_g}{C_c}$ ratios, namely $\frac{C_g}{C_c} = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.25, 1.5]$
4. Initial conditions for the un-adapted mesh are:
   (a) $c_l = 0.3429$; error of 1.82%
   (b) $c_m = -0.0385$; error of 1.64%
   (c) Computational cost of the unrefined mesh is 545.81 seconds if using a simple Jacobi solver [47].

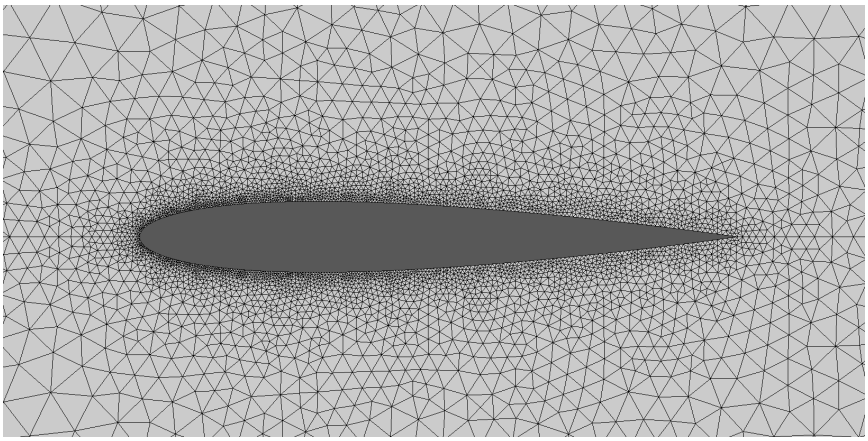

Fig. 5.6: Close-up of the un-adapted fine NACA0012 mesh.

Computational times for the various cases are reported in Table 5.3. The simulations were carried out on the UCT HPC's Dell C6145 racks. These house AMD Opteron 6274s at 2.2GHz and 2Gb of RAM at 1.3GHz. The typical pass time is presented for each batch and multiplied by the number of passes for a given batch to obtain the total refinement time. Total run time for each batch is given as an average in addition to the standard deviation (with the non-converging cases excluded). The typical refinement time is shown as a percentage of total run time and in the final column of the table, the increase in computational cost relative to the unrefined case is shown (due to repeated solves).

Tab. 5.3: Typical computational time for 2D fine mesh refinement reported in seconds

Batch Name	Pass Time	Tot. Refi. Time	Tot. Run Time	Std. Dev. Tot. Run	as % of Run Time	Inc. in Cost
5 LA	20.03	100.17	1 597.66	187.76	6.27	2.93
5 LD	16.86	84.30	1 586.91	228.82	5.31	2.91
10 LA	18.39	183.91	2 658.66	196.89	6.92	4.87
10 LD	16.13	161.33	2 395.13	372.60	6.74	4.39

For the above, the actual refinement computational cost was low and all runs used $< 10\%$ of full run times to perform the refinement. The increase in total CPU cost, when compared with the unrefined mesh, was well less than the number of passes applied. The increase in overall CPU time was significant, but not deemed prohibitive bearing in mind the considerable increase in accuracy achieved (Figures 5.7 and 5.8). To the reader it is noted

that this study served as a first assessment of the r-refinement technology, and little effort was applied to scheme optimization. It was found that the refinement threshold (flow solver convergence tolerance) had a significant effect on overall scheme efficiency, and should be investigated further.

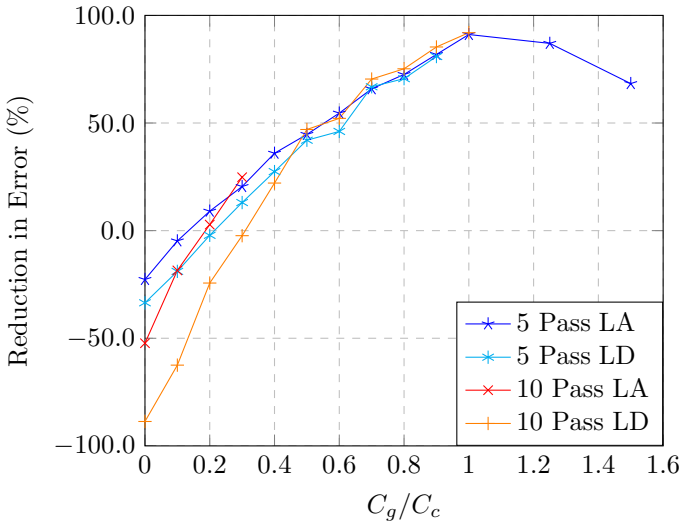

Fig. 5.7: Graph of the error reduction in the coefficient of lift vs. the C_g/C_c ratio on the fine NACA0012 mesh. Here, LA and LD respectively denote Laplacian smooth activated and deactivated.

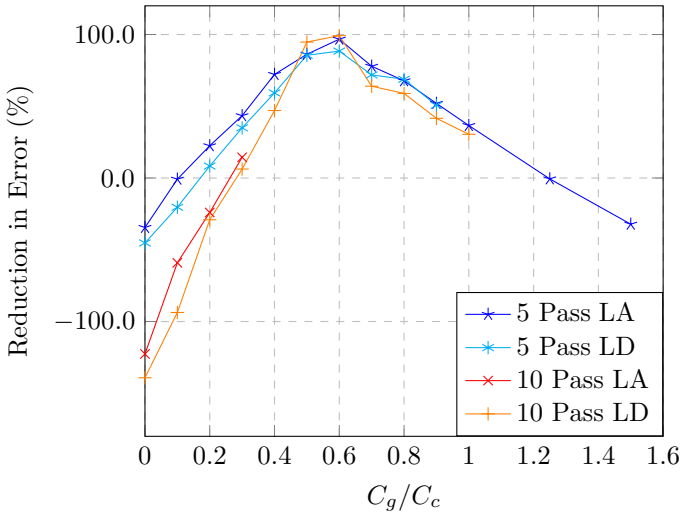

Fig. 5.8: Graph of the error reduction in the coefficient of moment vs. the C_g/C_c ratio on the fine NACA0012 mesh. Here, LA and LD respectively denote Laplacian smooth activated and deactivated.

Missing data points indicate tests for which the flow solver failed to re-converge (likely due to deterioration in element quality). As can be seen, this was more common for those batches with increased refinement passes as well as those with a higher gradient weighting. However, not one test failed to have the refined mesh reprocessed, inferring that no invalid elements were created (Elemental$^{\text{TM}}$ does checks for this).

The results obtained were, for the most part, satisfactory and a clear demonstration of the scheme's ability to significantly improve accuracy. Figure 5.9 shows the general trend of node movement, where nodes moved toward the aerofoil and further clustered around the nose, tail and, importantly, along the shock. For this case, the optimal C_g/C_c ratio is about 0.3 to 0.8 if an improvement in both c_l and c_m is sought. While increasing the weight of the gradient did seem to improve the lift coefficient, the chance of solver rejection was raised; and, more importantly, a clearly negative effect on the moment coefficient was observed. This again could be due to element quality deterioration around the leading and trailing edges and is to be investigated as further work. Finally, the effect of increasing the number of passes seemed to do little for improvement in accuracy and had a clear negative effect on solver acceptance. This is suggestive that r-refinement without element quality control has reached a limit.

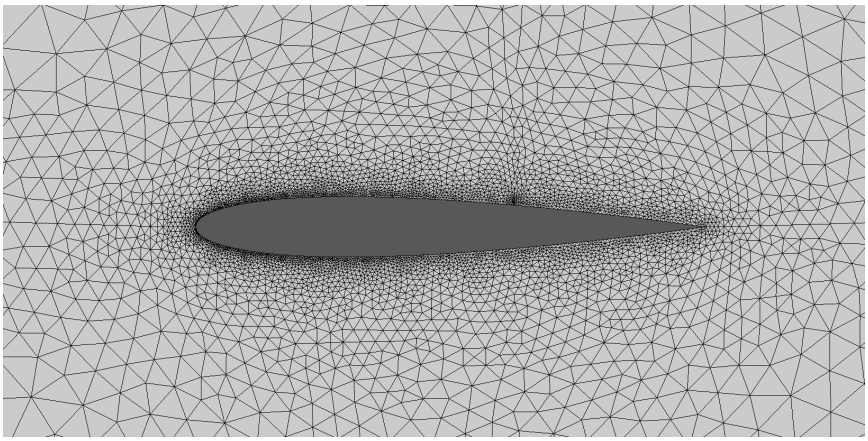

Fig. 5.9: Adapted mesh for test 10LD for a C_g/C_c ratio of 0.5.

Finally, the effects of the Laplacian smoother with regard to accuracy improvements are only noticeable for high curvature weighted monitor functions and high refinement passes. Figure 5.10 shows the difference in refinement around the shock. The importance of clustering nodes in the area of shock is important; however, the curvature monitor function tended to cluster nodes on either side. The Lapacian, when applied, relaxed this and so can be attributed with the improved accuracy in this area; however, little effect to solver acceptance was observed between using the smoother and not ($C_g/C_c = [0.3 : 0.5]$).

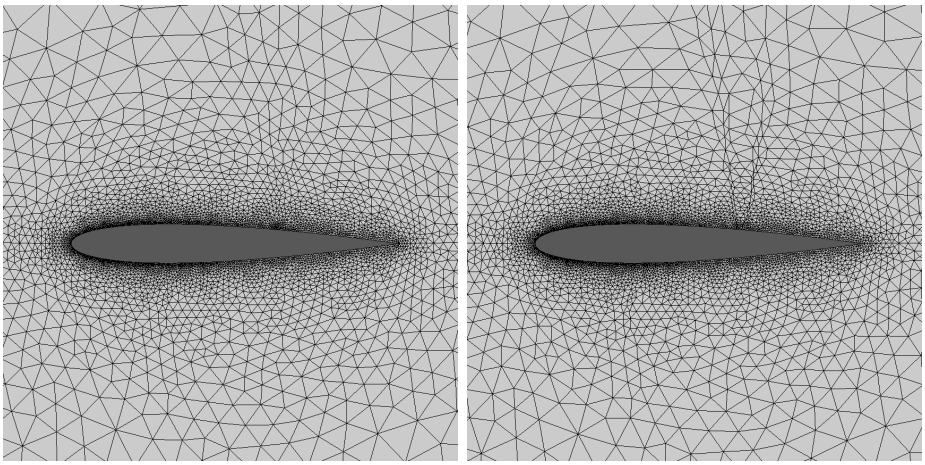

Fig. 5.10: Adapted meshes for test 10LA(left) and 10LD(right) for a C_g/C_c ratio of 0.0.

5.3.3 Coarse Mesh Refinement

The second mesh was significantly coarser than the previous 'fine' mesh. This example was perhaps the most critical since the refinement scheme needed to demonstrate applicability to test cases which were significantly far from an accurate solution. The set up for these tests was similar to that of the fine, with the exception of an absent smoother (as this was found to be of little value above). Instead, here the effect of re-setting $\bar{\boldsymbol{K}}^0$ at the beginning of each refinement pass was investigated. This was expected to give better linear approximations to the non-linear governing equations. The number of refinement passes was also reduced. This is due to the softness of this mesh and the clear negative effect of over-refinement on solver acceptance. Softness was attributed to the coarseness of the mesh which featured, on average, longer edge lengths and therefore softer springs.

The set up for the refinement runs follows with Figure 5.11 showing the un-adapted mesh:

1. Four batches consisting of two 3 and two 5 pass refinements are done.
2. One of the two sub-batches will have $\bar{\boldsymbol{K}}^0$ reset on every refinement pass, while the other will have $\bar{\boldsymbol{K}}^0$ set on the first pass only.
3. In each sub-batch, 13 individual tests are done with $\frac{C_g}{C_c} = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.25, 1.5]$
4. Initial conditions for the un-adapted mesh are
   (a) $c_l = 0.3116$; error of 7.47%
   (b) $c_m = -0.0298$; error of 21.32%
   (c) Computational cost on the unrefined mesh is 71.95 seconds.

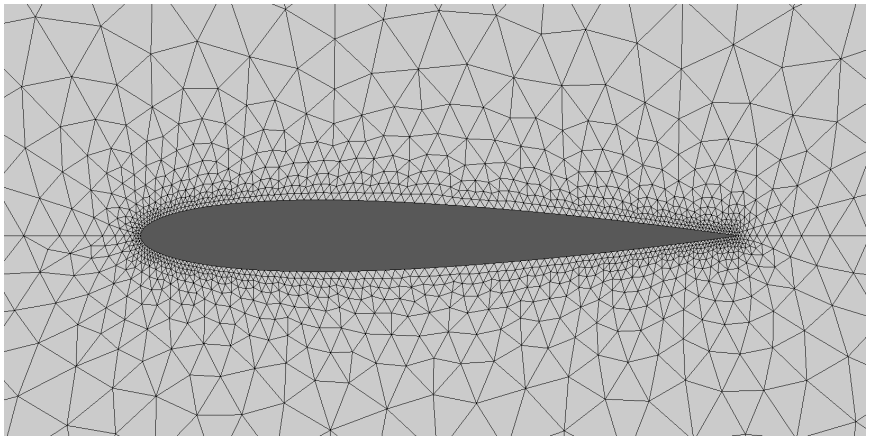

Fig. 5.11: Un-adapted coarse NACA0012 mesh.

Times for the computations are reported in Table 5.4 which were run on the same Dell C6145 cluster as previously done.

Tab. 5.4: Typical computational time for 2D coarse mesh refinement, reported in seconds.

Batch Name	Pass Time	Tot. Refi. Time	Tot. Run Time	Std. Dev. Tot. Run	as % of Run Time	Inc. in Cost
3 OE	1.43	4.30	111.18	12.14	3.87	1.55
3 OF	1.44	4.32	115.57	10.58	3.74	1.61
5 OE	1.63	8.16	159.97	16.53	5.10	2.22
5 OF	1.63	8.17	169.10	18.21	4.83	2.35

Refinement cost as a percentage of total time was small, which is consistent with results from the fine meshes. There was clearly little difference, cost-wise, in resetting the system stiffness matrix on every pass. Thus here, one would clearly favour resetting on every pass due to accuracy improvement, as is discussed shortly. Importantly, the increase in overall computational cost was significantly less vs. reduction in error (comparing both c_l and c_m) than previously seen. Weighing this against the large increases in accuracy (even though little has been done by way of scheme optimisation), significant value addition is deemed proven.

The graphs showing the reduction in error for the c_l and c_m from the base mesh follows. Here, the abbreviations OE and OF mean 'on every pass' and 'on first pass', referring to when $\bar{\boldsymbol{K}}^0$ was reset. Again, missing data points indicate tests which failed to re-converge and are not reflective of an invalid mesh (but rather deteriorating element quality).

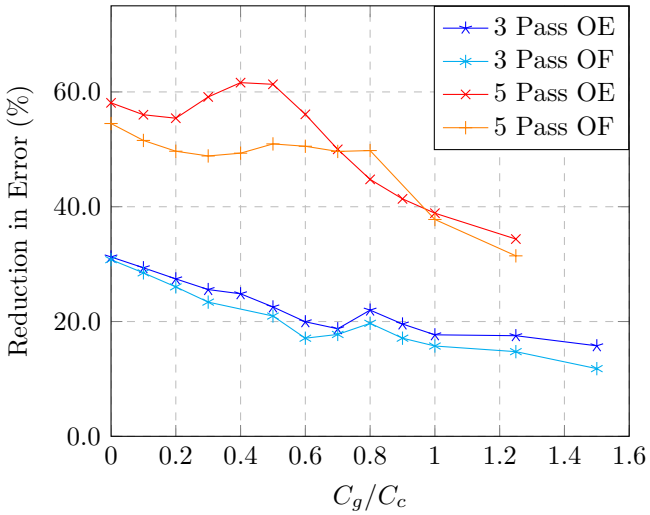

Fig. 5.12: Graph of the error reduction in the coefficient of lift vs. the C_g/C_c ratio on the coarse NACA0012 mesh. Here OE and OF respectively denote the resetting of $\bar{\boldsymbol{K}}^0$ on the every and first pass.

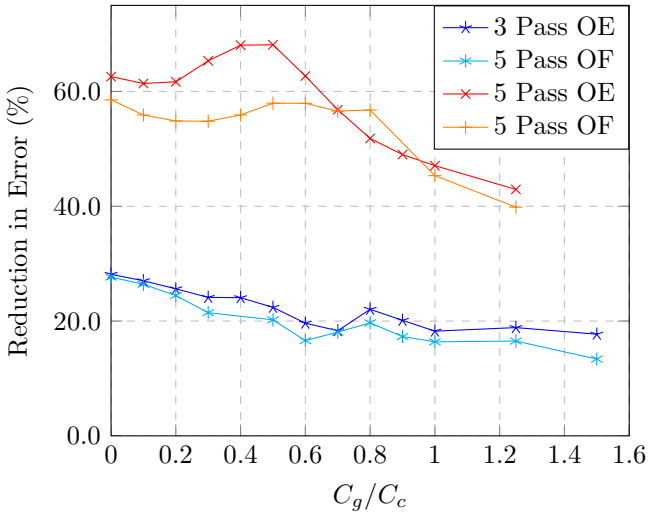

Fig. 5.13: Graph of the error reduction in the coefficient of moment vs. the C_g/C_c ratio on the coarse NACA0012 mesh. Here OE and OF respectively denote the resetting of $\bar{\boldsymbol{K}}^0$ on the every and first pass.

The results obtained confirmed those obtained on the fine mesh tests. There were clear consistencies between tests; a C_g/C_c ratio of 0.3 to 0.5 yielded the greatest improvement in accuracy for both c_l and c_m. Unlike

the fine results though, an increase in refinement passes yielded greater fidelity. As can be seen in Figure 5.14, the mesh nodes were pulled toward the aerofoil, and critically began to cluster around the shock. Better approximation of the non-linear system (by the resetting of $\bar{\boldsymbol{K}}^0$) had a significant effect on accuracy, especially over the optimal C_g/C_c ratios. Interestingly, the impact on lower passes was smaller; thus, the effect can be considered cumulative and the difference required multiple passes to exaggerate as one would expect.

One discrepancy between fine and coarse results was the trend of the c_l data. Unlike the fine mesh results, an increase in C_g/C_c ratio did not increase solution accuracy. The exact reason for this was unclear. Attributing factors could be initial mesh construction (Delaunay vs. paved) or initial mesh resolution. However, this discrepancy was not deemed too dire since the aggregate of data is suggestive of consistency in accuracy improvements in similar areas.

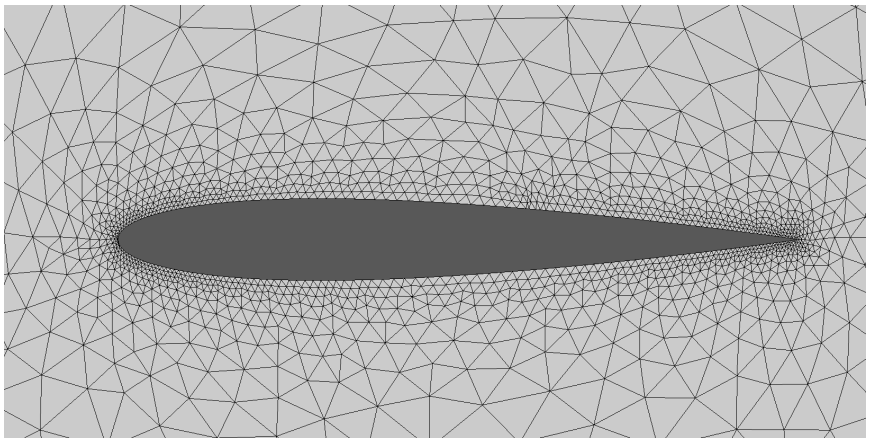

Fig. 5.14: Adapted meshes for test 5OE for a C_g/C_c ratio of 0.5.

5.4 3D FFAST Aerofoil Test Case

As with 2D, the primary goal of refinement in 3D was to demonstrate improvement in accuracy by adapting the mesh to field topology. The use of an industrially relevant aerofoil was also required and recent completion of the FFAST project provided such and aerofoil. Again, the curvature and gradient ratios for a density monitor function were investigated for improvements in accuracy in aerofoil characteristics. The Lapacian smoother was again abandoned since performance in 2D was not adequate to justify continued use. For these tests, the system stiffness matrix was reset on every refinement pass.

The flow was again inviscid, the angle of attack was set to 2.0° and

a Mach number of 0.8 was applied. Air properties were taken at cruise altitude. The calculation of the c_m number differs somewhat from the two dimensional case. The standard formulation in 3D, taken from [49], is of the form

$$c_m = \frac{M}{P_{dyn} A_{pl} l_{ch}} \tag{5.9}$$

where the dynamic pressure P_{dyn}, planform area A_{pl} and moments M are easily computed. The characteristic chord length l_{ch} was taken into account, as well as the swept back nature of the aerofoil (see Appendix D).

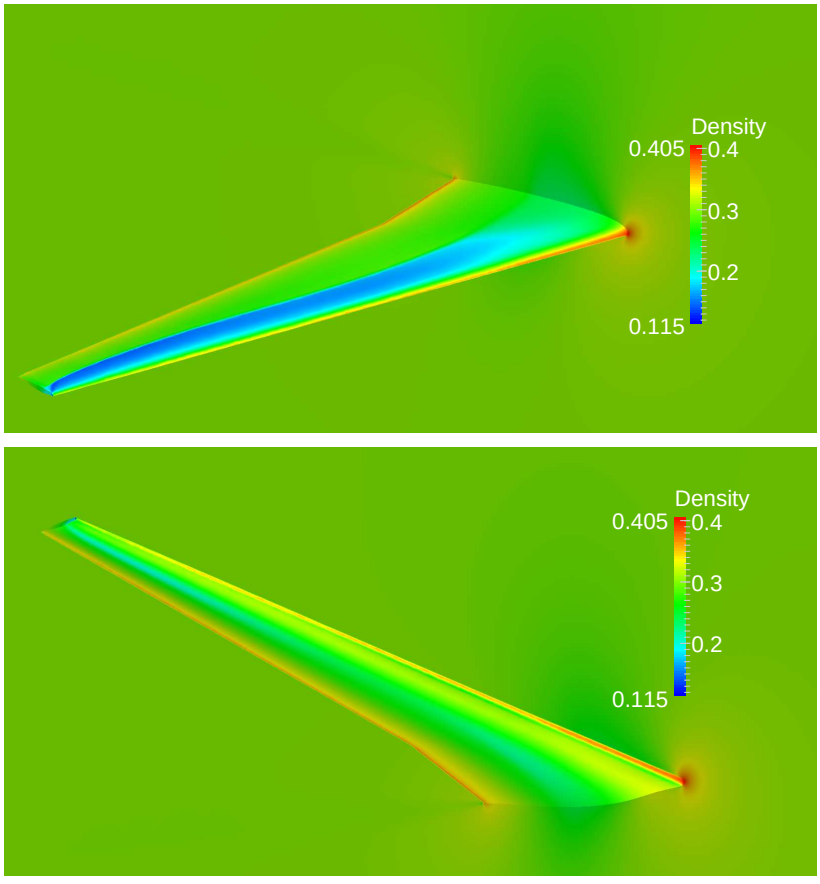

Fig. 5.15: Density field plot for the FFAST aerofoil as seen from above (top) and below (bottom).

The mesh independent solution was again as previously obtained. Figure 5.15 depicts the density field over the aerofoil on the finest mesh.

Tab. 5.5: Results from the various meshes for c_l and c_m on the FFAST aerofoil.

No.	Nodes	c_l	c_m
1	270 869	0.28856	0.20056
2	380 934	0.29081	0.20242
3	573 520	0.30116	0.21205

Tab. 5.6: Results from the mesh independence study for c_l and c_m in on the FFAST aerofoil.

	c_l	c_m
ζ_{extrap}	0.2876	0.1999
GCI	0.41 %	0.43 %

The grid used for the refinement study had a node count of 155k and was constructed from tetrahedrons. The refinement threshold was again set to $1.0e-3$. The set-up for the refinement was as follows:

1. Three batches consisting of 3, 5 and 7 pass refinements are done.
2. $\bar{\boldsymbol{K}}^0$ is reset on every refinement pass.
3. In each sub batch 13 individual tests are done with varying $\frac{C_g}{C_c}$ namely, $\frac{C_g}{C_c} = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.25, 1.5]$
4. Initial conditions for the un-adapted mesh are
   (a) $c_l = 0.27712$; error of 3.64%
   (b) $c_m = 0.1889$; error of 5.45%
   (c) Computational cost of the unrefined mesh is 6 hours 16 minutes.

Computational costs was found to have an overall increase in cost that was less than that of 2D, with no batch reported to have increased total cost by more than double (whether measured by time or flow solver iteration). Typical refinement costs constituted circa 30% of total run time, the increase due to the increased complexity of volume calculations (as now tediously noted the current algorithm has yet to be fully optimized).

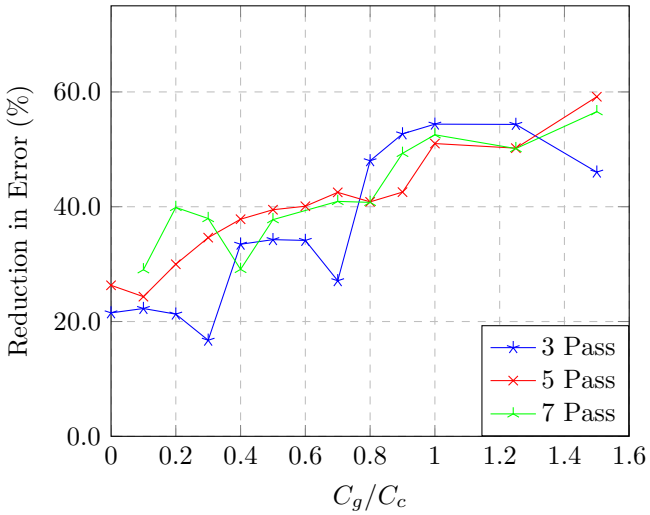

Fig. 5.16: Graph of the error reduction in the coefficient of lift vs. the C_g/C_c ratio on the FFAST aerofoil.

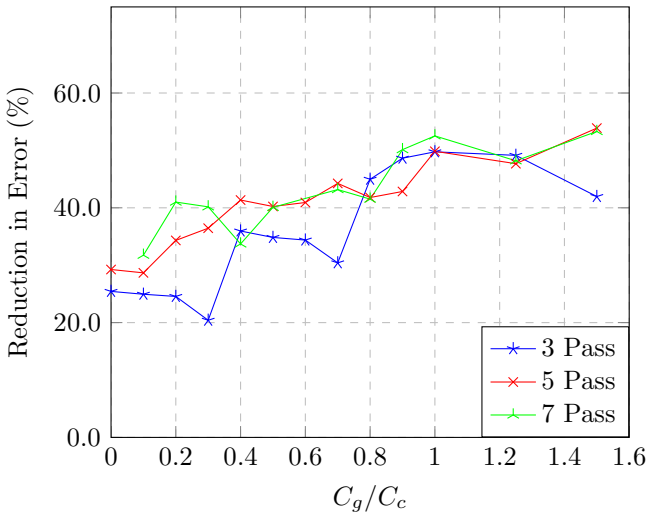

Fig. 5.17: Graph of the error reduction in the coefficient of moment vs. the C_g/C_c ratio on the FFAST aerofoil.

Reduction in errors, seen in Figures 5.16 and 5.17, peaked differently from the 2D test cases. Here reduction in error improved greatly around a C_g/C_c ratio of unity. At these ratios, improvements in accuracy peaked circa 50%. The relatively good performance of these ratios, differing somewhat from literature, was not to surprising after some examination. With the higher angle of attack, the transonic shock would have been stronger than

that of the 2D cases; thus, driving mesh adaptation via these gradients would naturally have tended to more accurately predict the c_l and c_m numbers.

5.5 3D Mesh Deformation

Mesh deformation testing was done primarily to demonstrate that the scheme could be used to significantly deform a mesh. To stress test this, an unrealistically large deformation was selected to demonstrate mesh validity past the point which would normally be observed in FSI simulations. A 200k node FFAST mesh was used as the base mesh to be deformed.

Euler-Bernoulli beam theory was employed to fictitiously deflect the wing which was modelled as a cantilever beam. In practice, linear beam theory would not be applicable to such large displacements. However, the objective was to demonstrate robustness of the mesh deformation under large deflections.

To effect the deflection, nodes were projected onto the beam, modelled as a plane running through the wing, prior to deflection. Using the projected position, the displacements were computed and prescribed as a Dirichlet condition to the truss network. For the mesh deformation, the following parameters were assumed for the cantilever beam used:

$$\frac{F}{EI} = 0.0015m^{-2}$$
$$L = 29.1m$$

For mesh movement purposes, the above displacement was broken into three equal steps. The resulting displacement is depicted in Figure 5.18. No invalid elements were created and the flow solver continued convergence. Tip displacement was recorded to be 18.1m, which is two-thirds of the span. The volumetric deformed mesh and a close up of the tip is shown in Figures 5.19 and 5.20 respectively. The calculation was conducted on a Intel i7 2.30GHz chip with 8Gb of DDR3 RAM at 800MHz. Computational time for the deformation was recorded at 16 minutes 52 seconds.

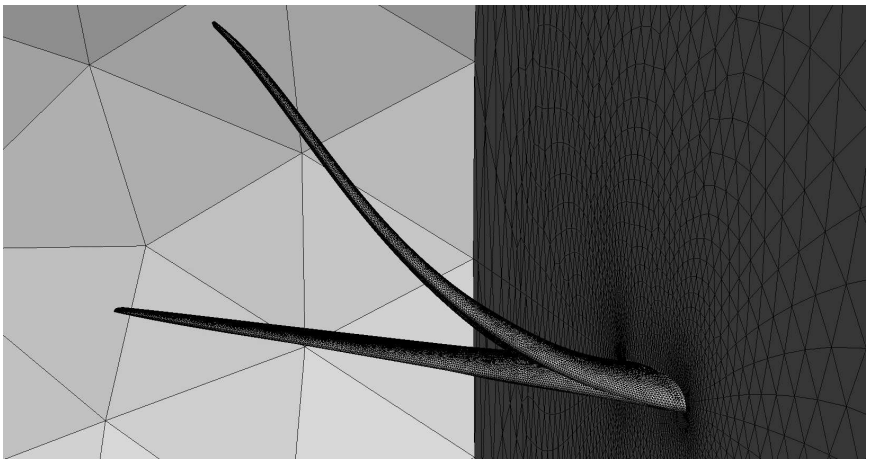

Fig. 5.18: Un-deformed and deformed FFAST aerofoil

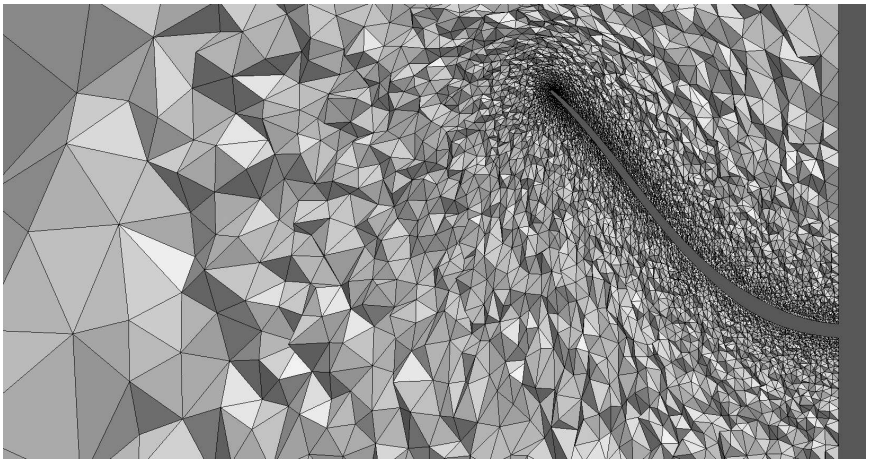

Fig. 5.19: Deformed volumetric mesh

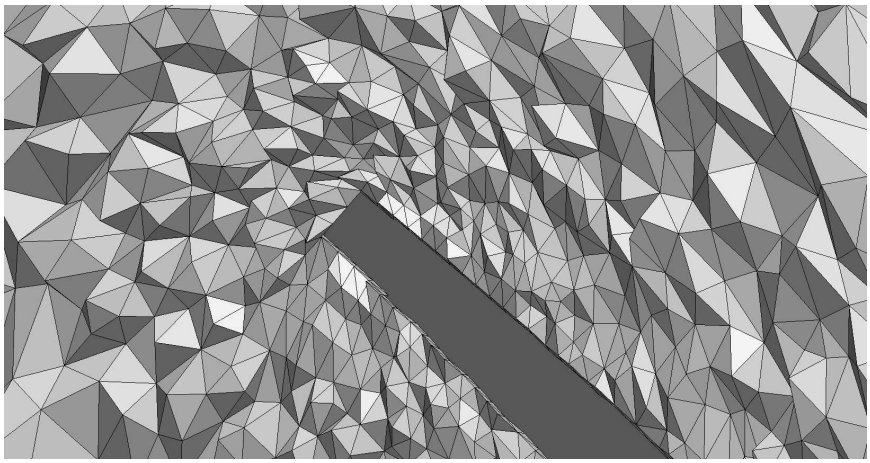

Fig. 5.20: Close-up of the deformed volumetric mesh around the aerofoil tip.

6 Conclusions and Recommendations

6.1 Summary and Concluding Remarks

If continued improvements to CFD computational efficiency (accuracy vs. computational cost) are to be sought, mesh adaptation will need to be improved. *A posteriori* knowledge of the flow field can aid in the adaptation process. While each type of adaptation (p-, h- and r-) has its limitations, in combination, improvements to efficiency can be obtained. Here, r-refinement was investigated with this purpose.

The ball-vertex truss networked variant was utilised as the mechanism through which to perform mesh adaptation. The method was selected for its robustness in mesh deformation and simplicity with regard to other schemes. Further, initial exploratory work had been carried out on the scheme's ability to reduce error on a computational mesh.

The formulation entails attaching fictitious springs to nodes, the stiffness of which are a function of the inverse edge lengths. Forces were applied to nodes using a forcing monitor function which was formulated using field gradients and curvature (a novel approach), which are indicative of discretization error. Together, these provide a balance between robustness and node relocation (which reduce error).

For mesh adaptation, industrial relevance dictates that complex geometries are refined, as well as that nodes must slide along the boundary. In most applications, geometric feature definition is not kept past mesh generation; thus, to obtain effective error reduction along boundaries, a Bèzier curve/surface optimization process was used to place nodes back onto the boundary.

The entire scheme was implemented inside Elemental$^{\text{TM}}$'s compressible flow solver. Novelty was sought in the discretization and solution of the truss governing equation by resetting the stiffness matrix at the beginning of every r-refinement pass. The effect is better approximation of the non-linear nature of the governing equations for each iteration.

After good performance in 2D analytical tests cases, the r-refinement

scheme was applied to the 2D NACA0012 aerofoil under transonic flow conditions. Here, the objective being reduction in error of the lift and moment coefficients, c_l and c_m, with respect to a mesh independence solution. Two tests were conducted: one on a fine mesh, the other course. For the fine mesh, error reduction peaked close to 100% and on the course mesh at around 65%. Reductions of at least 50% were obtained for a gradient-curvature ratio of circa 0.5 for all cases. Modest increases in computational time were recorded for the test, thereby resulting in good improvements to computational efficiencies. Scheme robustness was found to be good with no negative volume elements created.

Final testing was carried out on the FFAST aerofoil in 3D. Results had a slightly different trend, as greater reduction in errors were seen for increased ratios of C_g/C_c (circa unity). This is expected to be due to an increase in the angle of attack. Finally, a mesh deformation problem was successfully completed by way of an unrealistically large wing deflection and zero inverted elements.

6.2 Recommendations

Although this project was an initial step into r-refinement and served to inform future developments, the developed scheme proved effective and of industrial relevance. To further the work done to date, the following recommendations are made:

1. Further optimisations should be sought through a more intelligent refinement algorithm, such as a dynamic selection of pass number and refinement threshold.

2. An element quality metric and mesh cosmetics tool should be constructed to improve post refinement element quality.

3. Ortho-semi-torsional springs could be considered, as these appear (from visual inspection of papers) to also produce good element quality.

4. Monitor functions for different physics and flow phenomena should be further investigated.

Bibliography

[1] C. L. Bottasso, D. Detomi, and R. Serra. The ball-vertex method: a new simple spring analogy method for unstructured dynamic meshes. *Computational Methods in Applied Mechanics and Engineering*, 194:4244–4264, 2005.

[2] K. Morgan, O. Hassan, and R. Sevilla. O.C. Zeinkewicz Opening Lecture. In *4th African Conference on Computational Mechanics*, 2015.

[3] D. Franke, A. Duster, V. Nubel, and E. Rank. A comparison of the h, p, hp, and rp version of the FEM for the solution of the 2d hertzian contact problem. *Computational Mechanics*, 45.5:513–522, 2010.

[4] I. Babuska and W. C. Rheinboldt. Error estimates for adaptive finite element computations. *SIAM Journal on Numerical Analysis*, 15:1978, 1978.

[5] O. C. Zeinkewicz and J. Z. Zhu. A simple error estimator and adaptive procedure for practical engineerng analysis. *International Journal for Numerical Methods in Engineering*, 24:337–357, 1987.

[6] M. J. Aftosmis and M. J. Berger. Multilevel error estimation and adaptive h-refinement for cartesian meshes with embedded boundaries. In *40th AIAA Aerospace Sciences Meeting and Exhibit*, volume 863, 2002.

[7] F. Alauzet, P. L. George, B. Mohammadi, P. Frey, and H. Borouchaki. Transient fixed point-based unstructured mesh adaptation. *International Journal for Numerical Methods in Fluids*, 43:729–745, 2003.

[8] A. Arnold, O. T. Bruhns, and J. Mosler. An efcient algorithm for the inverse problem in elasticity imaging by means of variationalr-adaption. *Physics in Medicine and Biology*, 56:4239–4265, 2011.

[9] P. K. Jimack and R. M. Kirby. Towards the development of an hp-renement strategy based upon error estimate sensitivity. *Computers & Fluids*, 46:277–281, 2011.

[10] C. de Boor. Good approximation by splines with variable knots. ii. In *Conference on the numerical solution of differential equations*, pages 12–20. Springer, 1974.

[11] C. J. Budd, W. Huang, and R. D. Russell. Adaptivity with moving grids. *Acta Numerica*, 18:111–241, 2009.

[12] T. Tang. Moving mesh methods for computational fluid dynamics. *Contemporary Mathematics*, 383:141–174, 2005.

[13] M. J. Baines, M. E. Hubbard, and P. K. Jimach. Velocity-based moving mesh methods for nonliner partial differential equations. *Communications in Computational Physics*, 10:509–576, 2011.

[14] M. Ahamadi and O. G. Harlen. A lagrangian finite element method for simulation of a suspension under planar extensional flow. *Journal of Computational Physics*, 227:7543–7560, 2008.

[15] M. Kenamond, M. Bement, and M. Shashkov. Compatible, total energy conserving and symmetry preserving arbitarty Lagrangian-Eulerian hydrodynamic in 2d rz-cylindrical coordiantes. *Journal of Computational Physics*, 268:154–185, 2014.

[16] E. J. Caramana and M. J. Shashkov. Elimination of artificial grid distortion and hourglass-type mothions by means of lagrangian subzonal masses and pressures. *Journal of Computational Physics*, 142:521–561, 1998.

[17] K. Miller and R. N. Miller. Moving finite element i. *SIAM Journal on Numerical Analysis*, 18:1019–1032, 1981.

[18] K. Miller. Moving finite elements ii. *SIAM Journal on Numerical Analysis*, 18:1033–1057, 1981.

[19] M. Grajewski, M. Koster, and S. Turek. Mathematical and numerical analysis of a robust and effcient grid deformation method in the finite element context. *SIAM Journal on Scientific Computing*, 31:1539–1557, 2009.

[20] W. Huang and R. D. Russell. A high dimensional moving mesh strategy. *Applied Numerical Mathematics*, 26(1):63–76, 1997.

[21] W. Huang and R. D. Russell. Moving mesh strategy on a gradient flow equation for two-dimensional problems. *SIAM Journal on Scientific Computing*, 20:998–1015, 1999.

[22] H. D. Ceniceros and T. Y. Hou. An efficient dynamically adaptive mesh for potentially singular solutions. *Journal of Computational Phyisics*, 172:609–639, 2001.

[23] A. S. Dvinsky. Adaptive grid generation from harmonic maps on riemannian manifolds. *Jounal of Computational Physics*, 95:450–476, 1991.

[24] J. U. Brackbill and J. S. Saltzman. Adaptive zoning for singular problems in two dimensions. *Journal of Computational Physics*, 46:342–368, 1982.

[25] A. M. Winslow. Numerical solution of the quasilinear poisson equation in a nonuniform triangle mesh. *Jounal of Computational Physics*, 1:149–172, 1966.

[26] J. F Thompson, Z. U. A. Warsi, and W. C Mastin. *Numerical grid generation: foundations and applications*, volume 38. North-holland Amsterdam, 1985.

[27] Y. Brenier. Polar factorization and monotone rearrangement of vector-valued functions. *Communications on Pure and Applied Mathematics*, 44(4):375–417, 1991.

[28] L. A. Caffarelli. The regularity of mappings with a convex potential. *Journal of the American Mathematical Society*, 5:99–104, 1992.

[29] L. A. Caffarelli. Boundary regularity of maps with convex potential. *Annals of Mathematics*, 144:453–496, 1996.

[30] G. L. Delzanno, L. Chacn, J. M. Finn, Y. Chung, and G. Lapenta. An optimal robust equidistribution method for two-dimensional grid adaptation based on MongeKantorovich optimization. *Journal of Computational Phyisics*, 227:9841–9864, 2008.

[31] P.A. Browne, C. J. Budd, C. Piccolo, and M. Cullen. Fast three dimension r-adaptive mesh redistribution. *Journal of Computational Phyisics*, 275:174–196, 2014.

[32] C. J. Budd, M. J. P. Cullen, and E. J. Walsh. Monge-Ampere based moving mesh methods for numerical weather prediction, with applications to the eady problem. *Journal of Computational Physics*, 235:247–270, 2013.

[33] J. T. Batina. Unsteady euler airfoil solutions using unstructured dynamic meshes. *AIAA journal*, 28(8):1381–1388, 1990.

[34] F. J. Blom. Considerations on the spring analogy. *International Journal for Numerical Methods in Fluids*, 32:647–668, 2000.

[35] C. Farhat, C. Degand, B. Koobus, and M. Lesoinne. Torsional springs for two-dimensional dynamic unstructured fluid meshes. *Computer Methods in Applied Mechanics and Engineering*, 163:2314–245, 1998.

[36] C. Degand and C. Farhat. A three-dimensional torsional spring analogy method for unstructured dynamic meshes. *Computers and Structures*, 80:305–316, 2002.

[37] C. O. E. Burg. A robust unstructured grid movement strategy using three-dimensional torsional springs. In *34th AIAA Fluid Dynamics Conference and Exhibit*, volume 2529. AIAA, 2004.

[38] D. Zeng and C. R. Ethier. A semi-torsional spring analogy model for updating unstructured meshes in 3D moving domains. *Finite Elements in Analysis and Design*, 41:1118–1139, 2005.

[39] G. A. Markou, Z. S. Mouroutis, D. C. Charmpis, and M. Papadrakakis. The ortho-semi-torsional (OST) spring analogy method for 3D mesh moving boundary problems. *Computational Methods in Applied Mechanics and Engineering*, 196:747–765, 2007.

[40] N. Acikgoz and C. L. Bottasso. A unified approach to the deformation of simplicial and non-simplicial meshes in two and three dimensions with guaranteed validity. *Computers and Structures*, 85:944–954, January 2007.

[41] T. J. Lin, Z. Q. Guan, J. H. Chang, and S. H. Lo. Vertex-ball spring smoothing: An efficient method for unstructured dynamic hybrid meshes. *Computers and Structures*, 136:24–33, 2014.

[42] N. Acikgoz. *Adaptive and Dynamic Meshing Methods for Numerical Simulations*. PhD thesis, Georgia Institute of Technology, 2007.

[43] L. L. Bucciarelli. *Engineering Mechanics for Structures*. Dover Books on Engineering. Dover Publications Inc., March 2009.

[44] C. J. Roy. Strategies for driving mesh adaptation in CFD (invited). In *47th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition*. AIAA, January 2009 2009.

[45] P. I. K. Liakopoulos and K. C. Giannakoglou. Unstructured remeshing using an efficient smoothing scheme. In *European Conference on Computational Fluid Dynamics, ECCOMAS CFD 2006*, 2006.

[46] D. Jones. and A. Gaitonde. *Innovation for Sustainable Aviation in a Global Environment*, chapter Future Fast Methods for Loads Calculations: The 'FFAST' Project, pages 110–115. European Union, 2012.

[47] A. G. Mowat, A. G. Malan, L. H. van Zyl, and J. P. Meyer. Hybrid finite-volume reduced-order model method for nonlinear aeroelastic modeling. *Journal of Aircraft*, 51:1806–18012, 2014.

[48] P. J. Roache. Quantification of uncertainty in computational fluid dynamics. *Annual Review of Fluid Mechanics*, 29(1):123–160, 1997.

[49] J. D. Anderson Jr. *Fundamentals of Aerodynamics*, chapter Chapter 1, pages 24–25. Mc Graw Hill, 5th in si untis edition, 2011.

[50] Fluent Inc. *GAMBIT NEUTRAL FILE FORMAT*, 2006.

[51] D. J. Walton and D. S. Meek. A triangular G1 patch from boundary curves. *Computer Aided Design*, 28:113–123, 1996.

[52] B. Etkin and L. D. Reid. *Dynamics of Flight Stability and Control*, chapter Appendix C, pages 357–360. John Wiley & Sons, Inc, 1996.

APPENDIX

A Feature Definition and File Format

Boundary faces are defined as element faces to which only boundary nodes are attached. Clearly the dimension of the boundary face will always be one lower than the mesh dimension. See Figure A.1

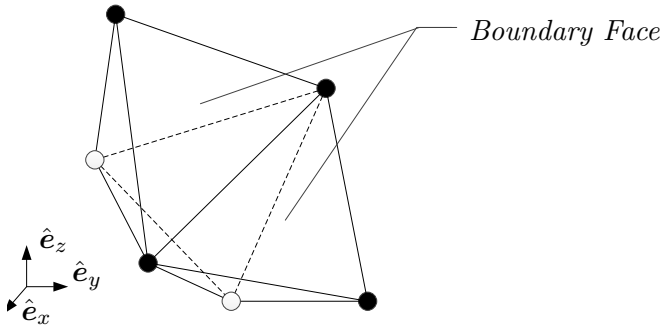

Fig. A.1: Three tetrahedral elements with two boundary elements on the boundary surface.

To ensure outward pointing normals, the crossing of vectors becomes dependent on the file format. In this investigation, GAMBIT neutral file formates were used, allowing calculation of face normals to be consistently outward pointing. More about the GAMBIT NUETRAL file format can be found in [50].

B Boundary Node Placement Procedures

The following placement procedures for boundary nodes apply to the six geometric placement types identified in Section 3.4.

Nodes on a smooth boundary are placed back onto the surface using Bèzier curve or surface fits, constructed from the boundary face information over which the node has moved.

Nodes on a straight line/flat plane (2D/3D) are moved similarly. In 2D, they are moved by only using that part of the displacement solution which is tangent to the boundary. Thus, the new co-ordinates for node i are given, using nomenclature from Figure 3.6 (top) by

$$\vec{x'}_i = \vec{x}_i + \vec{\delta x}_i \cdot \vec{t}_{ij_1} \tag{B.1}$$

For the 3D case the node is moved to i'_{pj} (using nomenclature from Figure 3.6 (bottom)) with Equation (3.7).

Nodes on straight ridges are displaced identically to the 2D straight line boundary movement. Again, Equation (B.1) can be used with $\vec{t}_{ii'}$ pointing to the appropriate connected ridge node.

Nodes on smooth ridges obtain their new co-ordinates in a two step procedure. The first step, $st1$, (Figure B.1) is to project the displacement solution $\vec{\delta x}_i$ onto the ridge plane which is coplanar with the ridge. This can be done using Equation (3.7). The plane normal $\vec{n}_{rp}$ formed by the boundary node normals $\vec{N}_i$ and $\vec{N}_j$.

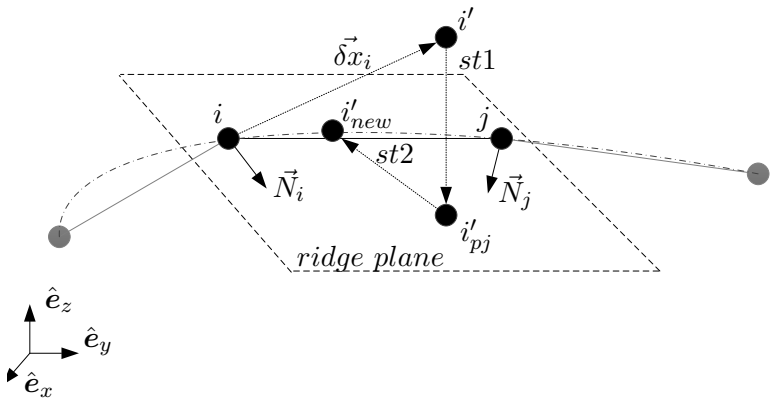

Fig. B.1: Steps taken to move nodes on curved ridges.

Step two ($st2$) involves moving point i'_{pj} to the new point on the ridge i'_{new}. This is done using a 2D Bèzier curve fit optimisation as previously described.

Nodes on corners have zero displacement and are ignored.
Nodes with an undesirable displacement solution are also ignored.

C Bezier Control Point Positioning

Bèzier curve and surface fits have long been used for many surface meshing tasks. These fits were employed for the task of placing boundary nodes back onto the boundary surface. The procedure for the construction of the Bèzier curves and surfaces follows from Walton and Meek [51].

In two dimensions a Bèzier curve D can be expressed as a polynomial:

$$D_e(l) = \sum_{i=0}^{p} B_{b_0,p}(l) V_{e,b_0}$$
$$l = [0:1] \tag{C.1}$$

The subscript e refers to the specific edge to which the curve belongs. This is necessary for 3D patches, explained shortly, where multiple curves are constructed. Here, V represents the control vertices for the curve and $B_{i,p}(l)$ is a Bernstein basis polynomials of degree p and is given by

$$B_{b_0,p}(l) = \binom{p}{b_0} l^{b_0} (1-l)^{p-b_0}$$
$$b_0 = [1:p] \tag{C.2}$$

where $\binom{p}{b_0}$ is a binomial coefficient. A typical Bèzier curve can then be drawn:

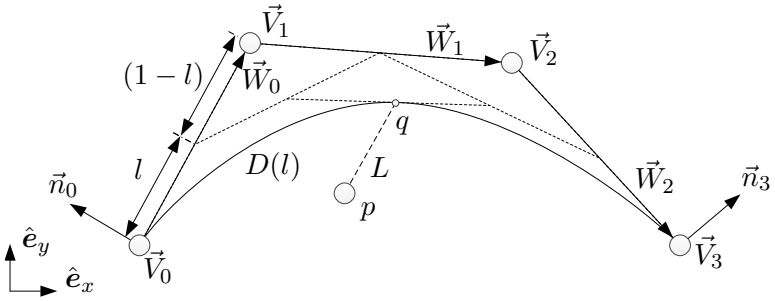

Fig. C.1: 3^{rd} Order Bèzier curve, with construction lines

Importantly, the curve $D(l)$ is orthogonal to the normals $\vec{n}_0$ and $\vec{n}_3$ at the two end control points and the directions of the normals dictate the placement of the intermediate control points V_1 and V_2. The orthogonality at the end points allows for a continuous surface to be drawn piecewise.

Point q represents a point on $D(l)$ where the construction geometry (drawn in dotted lines) is tangent to the curve and is thus given by

$$D'_e(l) = 3\sum_{b_0=0}^{p} B_{b_0,p-1}(l)\vec{W}_{e,b_0} \tag{C.3}$$

Where $\vec{W}$ is a vector connecting the relevant control points. Minimising the length of L with $D'(l)$ can be done such that the closest point on $D(l)$ can be found. This is the principle behind the optimisation process for node placement back onto the boundary curve.

For 3D, surface fits were used. To do this, a quadratic Bèzier patch S is created over 2D triangular element faces. The surface, described in barycentric co-ordinates and denoted w, is given by

$$S(w_1, w_2, w_3) = \sum_{b_1+b_2+b_3=4} P_{b_1,b_2,b_3} \frac{4!}{b_1!b_2!b_3!} w_1^{b_1} w_2^{b_2} w_3^{b_3} \tag{C.4}$$

$$w_1, w_2, w_3 \geq 0;\ w_1 + w_2 + w_3 = 1;\ b_1, b_2, b_3 \geq 0$$

Shown in Figure C.2 are the control point positions for a G^1 surface patch. Each edge is made up of a 3^{rd} order Bèzier Curve.

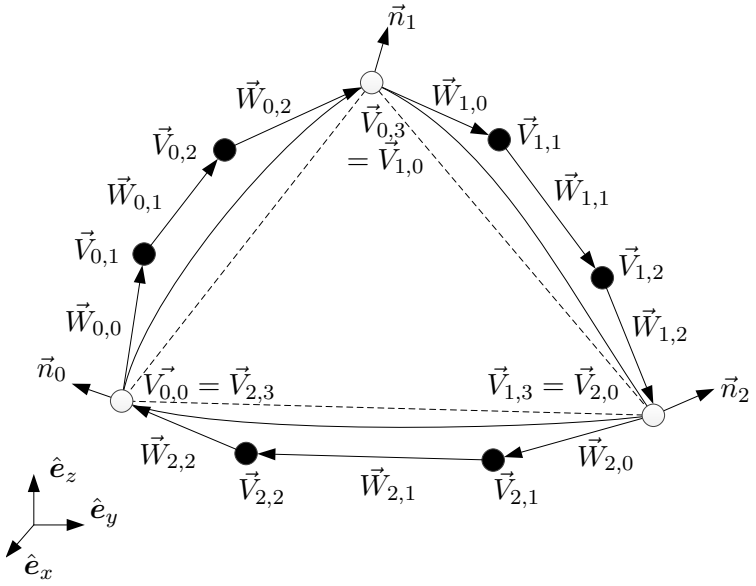

Fig. C.2: Triangular Domain with Cubic Bèzier Curve fits on each of the edges

The placement of the control points for each of the Bèzier Curves is given by,

$$\vec{V}_{e,1} = \vec{V}_{e,0} + \frac{d_e}{18}(6\vec{\Gamma}_e - 2\varrho_e\vec{n}_e + \sigma_e\vec{n}_{e+1}) \tag{C.5}$$

and

$$\vec{V}_{e,2} = \vec{V}_{e,3} - \frac{d_e}{18}(6\vec{\Gamma}_e + \varrho_e\vec{n}_e - 2\sigma_e\vec{n}_{e+1}) \tag{C.6}$$

The values for d_e, $\vec{\Gamma}_e$, ϱ_e and σ_e can be found in [51], and are functions of the end control points $\vec{V}_{e,0}$ and $\vec{V}_{e,3}$ and their normals.

Finally, the control points $\vec{P}$ for the quadratic surface fit can be found.

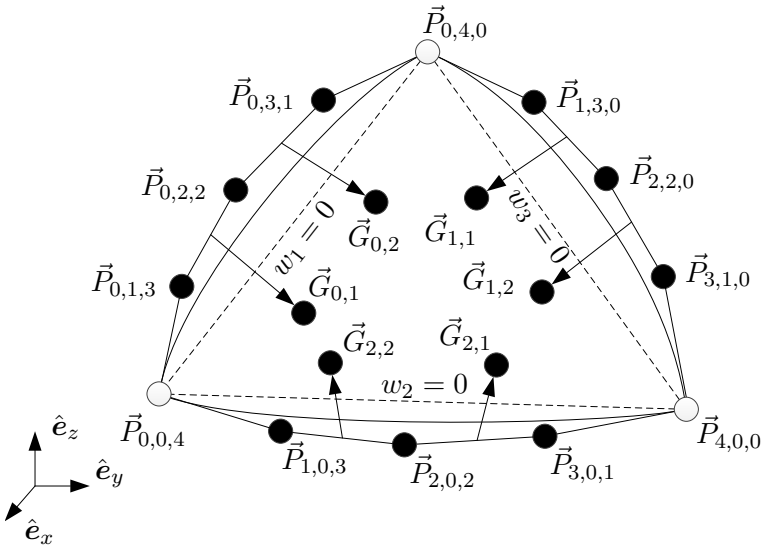

Fig. C.3: Control Points for a Gregory (G^1) patch

Using Figure C.3, the boundary control points are shown and can be obtained by,

$$\vec{P}_{0,b_4,4-b_4} = \vec{Q}_{0,b_4}, \vec{P}_{b_4,4-b_4,0} = \vec{Q}_{1,b_4}, \vec{P}_{4-b_4,0,b_4} = \vec{Q}_{2,b_4}$$
$$b_4 = 0....4 \tag{C.7}$$

with

$$\vec{Q}_{e,b_4} = \frac{1}{4}(b_4\vec{V}_{e,b_4-1} + (4-b_4)\vec{V}_{e,b_4}) \tag{C.8}$$

The remaining control points for the surface are given by

$$\vec{P}_{1,1,2} = \frac{1}{w_1 + w_2}\left(w_1\vec{G}_{2,2} + w_2\vec{G}_{0,1}\right) \tag{C.9}$$

with

$$\vec{G}_{e,1} = \frac{1}{2}\left(\vec{Q}_{e,1} + \vec{Q}_{e,2}\right) + \frac{2}{3}\lambda_{e,0}\vec{W}_{e,1} + \frac{1}{3}\lambda_{e,1}\vec{W}_{e,0} + \frac{2}{3}\mu_{e,0}\vec{A}_{e,1} + \frac{1}{3}\mu_{e,1}\vec{A}_{e,0} \tag{C.10}$$

Control point $\vec{P}_{1,2,1}$ and $\vec{P}_{2,1,1}$ can be found similarly to C.9. The determination of λ, μ and $\vec{A}$ from Equation (C.10) can again be found in [51]. The surface is now constructed with tangent ribbons along each of the edges ensuring a smooth surface over adjoining elements.

With all of the above, any point on the surface patch S can be found; thus, the length from an arbitrary point over the surface can be minimised with respect to the barycentric co-ordinates.

D FFAST Aerofoil Characteristic Chord Length Calculation

The aerofoil used was assumed to have a constant taper and sweep even though this is not strictly true. However, since the results are not to be compared to experimental data (for reasons given in previous chapters) and only for the mesh independence studies, such an assumption is justifiable. So long as assumption is applied consistently, results of improvement in accuracy should not be effected.

The taper ratio, Λ, is calculated as a ratio of the root and tip chord lengths. The mean chord length for a constant taper and sweep is calculated according to Equation (D.1), found in [52]

$$l_{ch} = \frac{2l_{ch,r}}{3}\frac{1+\Lambda+\Lambda^2}{1+\Lambda} \tag{D.1}$$

where

$$\Lambda = \frac{l_{ch,t}}{l_{ch,r}}$$

For the FFAST Aerofoil, the following values were obtained,

$$\Lambda = \frac{2.1230}{12.0615} = 0.1760$$
$$l_{ch} = \frac{2 \times 12.0615}{3}\frac{1+0.1760+0.1760^2}{1+0.1760} = 8.2528m$$