Real-time stress analysis of three-dimensional boundary element problems with continuously updating geometry

FOSTER, TIMOTHY,MARK

Real-time stress analysis of three-dimensional boundary element problems with continuously updating geometry

Timothy M. Foster

Thesis submitted towards the
degree of Doctor of Philosophy

Computational Mechanics Group
School of Engineering and Computing Sciences
Durham University
United Kingdom

June 2013

Real-time stress analysis of three-dimensional boundary element problems with continuously updating geometry

Timothy M. Foster

Abstract

Computational design of mechanical components is an iterative process that involves multiple stress analysis runs; this can be time consuming and expensive. Significant improvements in the efficiency of this process can be made by increasing the level of interactivity. One approach is through real-time re-analysis of models with continuously updating geometry. In this work the boundary element method is used to realise this vision. Three primary areas need to be considered to accelerate the re-solution of boundary element problems. These are re-meshing the model, updating the boundary element system of equations and re-solution of the system.

Once the initial model has been constructed and solved, the user may apply geometric perturbations to parts of the model. A new re-meshing algorithm accommodates these changes in geometry whilst retaining as much of the existing mesh as possible. This allows the majority of the previous boundary element system of equations to be re-used for the new analysis.

Efficiency is achieved during re-integration by applying a reusable intrinsic sample point (RISP) integration scheme with a 64-bit single precision code. Parts of the boundary element system that have not been updated are retained by the re-analysis and integrals that multiply zero boundary conditions are suppressed. For models with fewer than 10,000 degrees of freedom, the re-integration algorithm performs up to five times faster than a standard integration scheme with less than 0.15% reduction in the L_2-norm accuracy of the solution vector. The method parallelises easily and an additional six times speed-up can be achieved on eight processors over the serial implementation.

The performance of a range of direct, iterative and reduction based linear solvers have been compared for solving the boundary element system with the iterative generalised minimal residual (GMRES) solver providing the fastest convergence rate and the most accurate result. Further time savings are made by preconditioning the updated system with the LU decomposition of the original system. Using these techniques, near real-time analysis can be achieved for three-dimensional simulations; for two-dimensional models such real-time performance has already been demonstrated.

Declaration

The work in this thesis is based on research carried out in the Computational Mechanics Group, School of Engineering and Computing Sciences, Durham University. No part of this report has been submitted elsewhere for any other degree or qualification and it is all the work of the author unless referenced to the contrary in the text.

Parts of this work have been published in the following:

Journals

T. M. Foster, M. S. Mohamed, J. Trevelyan and G. Coates, "Rapid re-meshing and re-solution of three-dimensional boundary element problems for interactive stress analysis", *Engineering Analysis with Boundary Elements*, vol. 36, pp. 1331-1343, 2012.

Conferences

T. M. Foster, M. S. Mohamed, J. Trevelyan and G. Coates, "Real-time boundary element stress analysis", in *Proceedings of the 9th UK Conference on Boundary Integral Methods*, University of Aberdeen, July 2013.

T. M. Foster, M. S. Mohamed, J. Trevelyan and G. Coates, "Interactive boundary element analysis for engineering design", in *Proceedings of CM13*, Durham, UK, April 2013.

T. M. Foster, M. S. Mohamed, P. Fusco, J. Trevelyan and G. Coates, "Rapid re-solution of boundary element problems", in *Proceedings of the 10th World Congress on Computational Mechanics*, São Paulo, Brazil, July 2012.

T. M. Foster, M. S. Mohamed, P. Fusco, J. Trevelyan and G. Coates, "Rapid re-solution of boundary element problems", in *Proceedings of the 20th UK Conference of the Association for Computational Mechanics in Engineering*, The University of Manchester, March 2012.

T. M. Foster, M. S. Mohamed, J. Trevelyan and G. Coates, "Rapid re-meshing for real-time three-dimen-sional boundary element analysis", in *Proceedings of the 8th UK Conference on Boundary Integral Methods*, University of Leeds, July 2011.

M. S. Mohamed, T. M. Foster, J. Trevelyan and G. Coates, "Application framework and data structure for interactive stress analysis with the boundary element method", in *Proceedings of the 8th UK Conference*

on Boundary Integral Methods, University of Leeds, July 2011.

J. Trevelyan, G. Coates, M. S. Mohamed, T. M. Foster, S. K. Walker and S. H. Spence, "Towards interactive stress analysis for fatigue calculations for solid components", in *Proceedings of the 32nd Conference of the International Committee on Aeronautical Fatigue*, Montreal, Canada, May 2011.

T. M. Foster, M. S. Mohamed, J. Trevelyan and G. Coates, "Rapid re-meshing for real-time three-dimensional boundary element analysis", in *Proceedings of the 19th UK Conference of the Association for Computational Mechanics in Engineering*, Heriot-Watt University, Edinburgh, April 2011.

M. S. Mohamed, T. M. Foster, J. Trevelyan and G. Coates, "Application framework and data structure to introduce real-time stress analysis into early design stages", in *Proceedings of the 19th UK Conference of the Association for Computational Mechanics in Engineering*, Heriot-Watt University, Edinburgh, April 2011.

Acknowledgements

This project was conceived by my PhD supervisor, Professor Jon Trevelyan, and would never have been completed without his continuing enthusiasm and support. Along with Dr. Graham Coates, he has always been there to suggest ways to improve my work. Dr. M. Shadi Mohamed has somehow managed to motivate me and to keep me positive throughout my PhD, whenever the going got tough.

Thank you to the UK Engineering and Physical Sciences Research Council (EPSRC) for generously funding my research and to BAE Systems and Jesmond Engineering for their contributions. Dr. Stuart Spence of BAE Systems and Simon Walker of Jesmond Engineering have also generously provided regular feedback and support throughout this project, helping to steer and inform much of my work.

A huge thank you must also go to my parents, Carol and John Foster, my sister, Naomi Foster and my partner, Charlie Perkins, who have kindly proofread my entire thesis, despite not understanding a large portion of the work.

I am grateful to the many friends, both new and old, who have always been there to support and, when necessary, distract me throughout my time in Durham. Whilst working in the theatre, playing in bands, climbing mountains or just hanging out they have somehow managed to keep me sane.

Finally I would like to say a further thank you to Charlie Perkins. Her support throughout my PhD has been invaluable, giving me someone to bounce ideas off or to listen whilst I get them clear in my own head. She has always been there to prop me up, listen to my complaints, praise my (occasionally mad) ideas and to keep me going through thick and thin.

Tim Foster
Durham, June 2013

Contents

Figures

Tables

Algorithms

Acronyms

ACA	Adaptive cross approximation
BCG	Bi-conjugate gradient
BE	Boundary element
BEM	Boundary element method
BiCGSTAB	Bi-conjugate gradient stabilised
BIE	Boundary integral equation
CA	Concept Analyst
CA3D	Concept Analyst 3D
CAD	Computer aided design
CGS	Conjugate gradient squared
CPU	Central processing unit
CTL	Cantilever
CUBLAS	CUDA basic linear algebra subprograms
CUDA	Compute unified device architecture
DOF	Degrees of freedom
E-POD	Eigenvector based proper orthogonal decomposition
FE	Finite element
FEM	Finite element method
FMM	Fast multipole method
GMRES	Generalised minimal residual
GPU	Graphics processing unit
GUI	Graphical user interface
KLD	Karhunen-Loève decomposition
LU	Lower-upper
LUT	Look-up table
MINRES	Minimal residual
MSRS	Matrix system reduction solution
POD	Proper orthogonal decomposition
PWH	Plate with hole
QMR	Quasi-minimal residual
ROM	Reduced order modelling
RISP	Reusable intrinsic sample point
SVD	Singular value decomposition
SVD-POD	Singular value decomposition based proper orthogonal decomposition
TFQMR	Transpose free quasi-minimal residual
TWC	Thick walled cylinder

Nomenclature

For convenience the nomenclature has been split into sections. 'The mesh and geometry' lists symbols relating to the model and mesh geometry along with variables used in the meshing and re-meshing schemes. 'The boundary element method' contains symbols used in the boundary element method and for integration of the problem. 'Adaptive cross approximation' contains variables used only in the construction of an adaptive cross approximation of the system matrix. 'System solution' lists symbols used in linear and reduction based solvers and symbols related to the output such as stresses and error measures.

The mesh and geometry

a, b, c	Local coordinate axis system.
A	Area inside an element.
C	Centre of curvature.
$C_{\min}$	Minimum number of elements around any arc.
D	A diameter.
d	The distance through which updated geometry has been moved.
E_i	An element.
F	A flag.
F_{Eu}	Element update flag.
F_{Lu}	Line update flag.
F_{Nu}	Node update flag.
F_{Pu}	Patch update flag.
h	Normal distance from patch to a point.
K	Location of the tip of a cone.
l	Length of a line.
l_i	A distance along a line.
L	Length of an element.
$L_{\min}$	Shortest dimension of a quadrilateral element.
$L_{\max}$	Longest dimension of a quadrilateral element.
$\hat{L}_{\max}$	Nominal maximum element size.
L_i	A line.
M	Point on an axis or rotation.
n_E	Number of elements in the model.
n_{Eu}	Number of elements updated after a geometric perturbation.
n_N	Number of nodes.
n_R	Number of rings of points around arcs and return angles.
N	Unit outward normal to a surface.
N_i	A node.
O	Origin of a local coordinate axis system.
P	Point on a surface.
P_i	A patch.
Q	Quality of an element.
Q_A	Element quality according to the aspect ratio.
Q_C	Element quality according to the circle ratio.
Q_L	Element quality according to the length ratio.
Q_θ	Element quality according to the angle ratio.
$Q_{\min}$	Nominal minimum element quality.
$\bar{Q}$	Mean element quality.
$\bar{Q}_{\min}$	Nominal minimum mean element quality.
r	A radius.
R_c	Radius of a circumcircle.
R_i	Radius of an incircle.

$[R]$	Rotation matrix.
s	Semiperimeter of an element.
$\tilde{S}$	Standard deviation of element quality.
$\tilde{S}_{\max}$	Nominal maximum standard deviation of element quality.
t_m	Time to re-mesh a model.
T	A translation vector.
T_L	Translation vector applied to a line.
T_N	Translation vector applied to a node.
T_V	Translation vector applied to a vertex.
u, v, w	Parametric coordinates.
V_i	A vertex.
x, y, z	Cartesian coordinates.
α	Half angle in the tip of a cone.
β	The angle subtended by the material at a point.
γ	Mesh update propagation coefficient.
δ_1, δ_2	Start and end element lengths on a line.
θ	Angle about an axis of rotation.
$\theta_a, \theta_b, \theta_c$	Internal angles of a triangular element.
κ	Mesh refinement factor.
λ	Grading factor of elements along a line.
$\lambda_{\min}$	Minimum grading ratio along edges.
$\lambda_{\max}$	Maximum adjacency ratio around vertices.
ϕ	Angle between the outward normal of a patch and the z axis.
φ	Parametric internal angle of a conical surface.

The boundary element method

a_1, a_2	Tangential vectors.
b_1, b_2, b_3	Local orthogonal unit axes.
$[A]$	System matrix.
b	Body force per unit volume.
$\{b\}$	Right hand side vector.
$[B]$	Right hand side matrix.
c_{ij}	Smoothness coefficient.
D_{kij}	Derivative of U_{ij}.
e	Prescribed integration tolerance.
e_i	A unit vector in the direction i.
E	Young's modulus.
$[G]$	An influence matrix.
$[H]$	An influence matrix.
J	The Jacobian.
m	The number of Gauss points along each axis of a parametric element.
$\hat{m}$	The total number of Gauss points on an element.
n	Outward normal of a surface.
N_k	A shape function.
p	A source point.
q	A field point.
r	A distance between two points.
R	Distance between a point and an element.
S	Speed up factor.
S_u	Speed up of the update routine.
S_{kij}	Derivative of T_{ij}.
t	A traction.
$\{t\}$	Traction vector.
t^*	A weighting function related to the traction field generated from some arbitrary loads.
$\tilde{t}$	A prescribed traction boundary condition.
t_{ui}	Time to re-integrate a model during update step i.
T_{ij}	Traction kernel.
u	A displacement.
$\{u\}$	Displacement vector.
u^*	A weighting function related to the displacement field generated from some arbitrary loads.
U_{ij}	Displacement kernel.
w	A weight.
$\{x\}$	Solution vector.
$\{y\}$	Boundary condition vector.
Γ	A boundary.
δ_{ij}	Kronecker delta.
ε	A strain.
ϑ	Angle between ξ and η.
λ	Order of a singularity.
μ	Shear modulus.
ν	Poisson's ratio.
ξ, η	Parametric coordinates.

σ	A stress.
ψ	Scaling factor.
Ω	A domain.

Adaptive cross approximation

$[A']$	An admissible block of matrix $[A]$.
$\{b'\}$	Part of vector {b} associated with block $[A']$.
C	A set of points.
d_C	Maximal distance between any two points in C.
$\{e_i\}$	A unit vector in the direction i.
m'	Number of columns in an admissible block.
n'	Number of rows in an admissible block.
$r_{C_1C_2}$	Minimum distance between groups of points C_1 and C_2.
s	Number of vectors used in a cross approximation.
$[S_k]$	A cross approximation of a block reconstructed from k basis vectors.
$\{u_k\}$	Row vectors in a cross approximation.
$[U]$	Row vectors used in a cross approximation.
$\{v_k\}$	Column vectors in a cross approximation.
$[V]$	Column vectors used in a cross approximation.
γ	Scaling factor used to construct a cross approximation.
ε_F	An error calculated using the Frobenius norm.
κ	Block admissibility scaling factor.
$\imath, \jmath$	Arrays of IDs for rows/columns already used in a cross approximation.

System solution

$[A_i]$	System matrix after update i.
$\{b_i\}$	Right hand side vector after update i.
K_t	Stress concentration factor.
$[L]$	A lower diagonal matrix.
$[LU]$	LU decomposition matrices stored in a single matrix.
m	Number of basis vectors.
$[M]$	A preconditioning matrix.
n	Number of DOF/size of a system of equations.
$\hat{n}$	Maximum number of solver iterations.
n_u	Number of updated DOF.
n_{ui}	Number of DOF changed during update step i.
$\{r\}$	Residual vector.
s	A number of analysis runs.
S_s	Speed up of the solver routine.
$[S]$	Schur compliment.
t	Total re-analysis time.
t_B	Time to construct a set of basis vectors.
t_{si}	Time to solve a linear system of equations during update step i.
$[U]$	An upper diagonal matrix.
$\{\hat{x}\}$	Benchmark $\{x\}$ vector.
$\{x_i\}$	Solution vector after update i.
α	An eigenvalue.
ε_x	The error in vector $\{x\}$.
ε_σ	The error in vector $\{\sigma\}$.
$\{\zeta\}$	A vector of coefficients ζ_j.
κ	The condition number of a matrix.
$\{\sigma\}$	A stress vector.
$\{\hat{\sigma}\}$	Benchmark stress vector.
σ_t	Tangential stress.
σ_{VM}	von Mises stress.
$\{\varphi_i\}$	A single basis vector.
$[\Phi]$	An orthonormal basis.
$\{\psi_i\}$	A single basis vector.
$[\Psi]$	An orthonormal basis.

Chapter 1

Introduction

"The trouble with having an open mind, of course, is that people will insist on coming along and trying to put things in it."

Terry Pratchett

1.1 Thesis statement

The primary aim of this work is to build a new stress analysis computer aided design (CAD) software package that is capable of accurately re-analysing three-dimensional computer models of simple mechanical components in real-time as the geometry is updated by the user. This software has become known as *Concept Analyst 3D (CA3D)* after *Concept Analyst (CA)* [1], a two-dimensional boundary element (BE) CAD package which already features real-time functionality. It is intended that this software should be used at the conceptual design stage to aid the engineer in optimising the design of components or checking the tolerances. This would make it possible to design components more appropriately for their expected loading from the earliest design stages and thereby remove the need to make costly adjustments later in the design process if the geometry were found to be unsuitable. As the software is for conceptual purposes it should be intuitive to use.

To realise these goals in three dimensions, new boundary element techniques which incorporate a new re-meshing scheme and a modified re-integration algorithm, have been developed. These new algorithms enable re-use of unchanged model geometry, after a geometric perturbation, to limit propagation of changes through the mesh and thereby accelerate the reconstruction of the boundary element system of equations. Research carried out as part of this work has established that the updated system is most efficiently and accurately solved using a generalised minimal residual (GMRES) solver. The GMRES algorithm has been optimised for the current work and incorporates an approximate complete lower-upper (LU) preconditioner, generated during a full initial analysis.

In this work the term 're-analysis' has been used to define the process of updating the BE system of equations after a geometric perturbation, re-using unchanged parts of the mesh and hence unchanged matrix coefficients where possible, and solving the modified system thereby created.

This work has produced software capable of re-meshing and re-analysing small models (in terms of number of elements) up to ten times per second and has significantly accelerated the re-analysis of larger problems. This software can be used to rapidly evaluate stress concentrations and interactions for geometry which may be hard to find in handbooks such as Peterson's Stress Concentration factors [2].

1.2 Significance

Stress analysis of mechanical components has become an essential part of the validation of engineering designs but it can be time consuming and expensive. It is desirable to shorten the design cycle, thus reducing development costs and enabling new products to be brought to market in the shortest time possible. It is of particular importance to obtain the most suitable design at the conceptual design stage as the cost of any change at later stages grows rapidly. For many components this can be done only through rapid computational analysis of a wide range of initial geometries. This work aims to create a CAD software package capable of providing real-time feedback whereby the early design can be driven by the stress distribution in the model rather than simply validated later in the process.

This work could also be applied to other applications. It could be used to provide accurate real-time feedback to surgeons during training exercises or real operations. By coupling the analysis to an optimisation algorithm, efficient structures could be rapidly created for any application ranging from bridges to thermal conductors. The algorithms could also be used within graphics software to provide deformable models which can be interacted with, such as structures in computer games.

1.3 Background

Writing in 1990, Kane *et al.* [3] found no published work on iterative re-analysis using the boundary element method (BEM) and noted that many published finite element method (FEM) re-analysis techniques were slower than an entirely new analysis. Real-time analysis of two-dimensional models is now a reality [1]. However, real-time stress analysis of three-dimensional objects presents numerous additional challenges. Over the last decade, various FEM based schemes that aim to provide interactivity have been presented [4–7]. Meier *et al.* [8] review a range of deformable models that utilise both finite element (FE) and BE techniques. The BEM is a natural method to use where re-meshing is involved as changes need to be applied only to elements on the surface of the model. If the volume were meshed, as in the case of the FEM, then changes would propagate further through the model and many more degrees of freedom (DOF) would be affected. Mackie [4] presents a substructuring approach that attempts to ameliorate these difficulties. Wang *et al.* [9, 10] explore the potential of the BEM to generate real-time responses to forces applied to an elastic deformable object. They specifically review these techniques for application in providing haptic feedback for neuro-surgical simulations but provide a comprehensive review of techniques for deformable models using various methods, not just computational continuum mechanics. However, most of these are concerned with FE implementations as very few BE based strategies exist. Bro-Nielson and Cotin [11] were among the first to consider application of FE analysis to real-time simulation of surgical procedures. However, they limited analysis to a standard set of models for which the computationally expensive matrix inversions required to solve the problem have already been computed.

Real-time updating is also of interest to computer game developers to provide realistic deformable objects. For example, James and Pai [12] discuss the use of the BEM to provide physically accurate simulation of three-dimensional objects. However, as the technique is used purely for visual approximation of deformations, a much lower degree of accuracy is required than for stress analysis and a coarse mesh can be used resulting in a very fast analysis. Pre-computed solutions are also utilised. Deformable models for graphical display are extensively reviewed by Gibson and Mirtich [13] and Meier *et al.* [8], who are especially concerned with techniques applicable in surgical simulation, covering both FE and BE methods. Meier *et al.* classify methods based on the compromise between interactivity and motion realism, rating the BEM as a particularly good method due to computational speed and the fact that it is more robust than the other models.

CA [1], a two-dimensional BE stress analysis package that features real-time functionality, has already been developed. The current work aims to extend this interactive, real-time stress analysis capability into the three-dimensional domain. This involves the development of innovative techniques to generate, update and analyse the mesh. A key part of achieving this goal is the creation of a re-meshing algorithm. A high quality initial mesh must be created to produce an accurate initial analysis using the minimum number of elements. The mesh should further be capable of absorbing some of the distortion introduced as the user updates the model. A concise data structure is also required to aid rapid re-meshing.

Many authors have proposed algorithms for re-meshing FE and BE models, often in an adaptive scheme. Adaptive refinement schemes, which are extensively reviewed for BEs by Kita and Kamiya [14], inform re-meshing through the errors in an initial analysis. Depending on the scheme selected different strategies may be employed: h-refinement requires subdivision of an existing mesh; p-refinement retains the existing mesh but alters the order of the element; r-refinement maintains the same number of elements but requires regeneration of the entire mesh. These schemes may be used in combination, but do not usually allow for geometrical changes. If the geometry of a model is changing, it is possible to refresh only affected areas of the mesh. This has previously been applied to fluid flow problems using the h-refinement technique with the FEM [15]. However, some of the methods employed could easily be adapted for BE analysis.

It is important to minimise the number of elements updated during re-meshing (Trevelyan *et al.* [16] have shown this to be the major factor in reducing re-analysis response time, as illustrated in Figure 1.1). Reducing the number of updated elements gives rise to a reduction in both the time to re-integrate and re-assemble the linear system of equations generated by the BEM and the time required to solve these equations [17]. Michler [18] uses techniques based on radial basis functions to locally distort an aircraft mesh based on geometric changes. This approach is designed for complex geometry undergoing significant geometric perturbation. A simple, fast algorithm has been implemented for the geometries found in the current work. However, Michler's approach would be appropriate should more complex models be considered.

An iterative linear solver, for example GMRES [19], can be employed to accelerate the re-analysis beyond anything achievable with a direct solver. Such iterative methods have already been applied to the BEM [20–22] and have been shown to be effective in accelerating the solution. However, these previous works do not achieve the speed necessary for real-time re-analysis. As the majority of the mesh remains unchanged when the model geometry is updated, the results of the initial solve can be used to provide both a suitable initial estimate of the solution and an appropriate preconditioning matrix that can be used to accelerate solution of the evolving system matrix.

Another approach is to use model order reduction to reduce the size of the problem by expressing it in a compressed form. Linear solvers that utilise model order reduction have been developed by Leu [23], Amsallem and Farhat [24], Ryckelynck *et al.* [25] and Kerfriden *et al.* [26]. A trivial solve time can be achieved by applying these techniques. However, significant overheads are required to construct and update the reduced system.

For large problems the solution of the BE system of equations dominates the solution time [27]. However, these large problems are not amenable to real-time analysis due to current hardware limitations. This work will therefore concentrate on the solution of small problems. For small systems the solution no longer dominates and methods to accelerate the construction of the system must be considered.

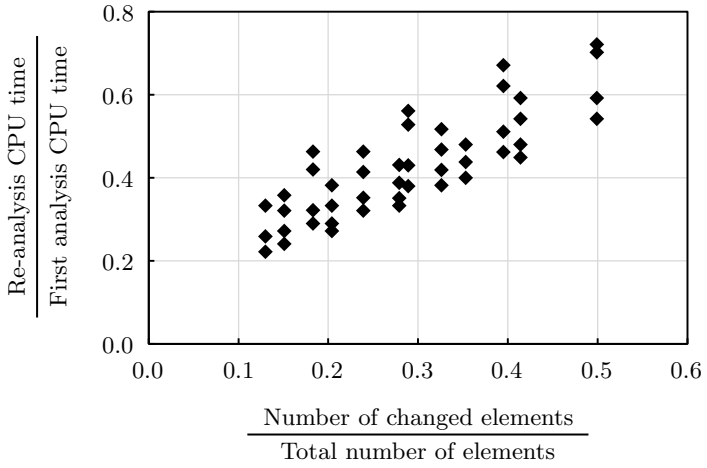


Figure 1.1: Re-analysis performance for three-dimensional applications (Reproduced from [16]).

1.4 The boundary element method

The BEM is a standard method of analysis in the solution of partial differential equations, and is the subject of numerous texts such as books by Becker [28], Kane [29] and Trevelyan [30].

Trevelyan [30] presents a good introduction to the BEM which establishes the basics of the method and is easy for engineers who are new to the technique to follow. However, he does not explore the process in great mathematical depth or give enough detail for comprehensive analysis. For more in depth detail on the mathematics behind, and the implementation of, the BEM, the author recommends Kane [29] and Becker [28]. A summary of the history of the BEM can be found in Section 2.1 and a derivation of the method is given in Section 2.2 of this thesis.

As has already been stated, the BEM is a natural method to use where re-meshing is involved as changes need to be applied only to elements on the boundary of the model, there are no internal elements. However, there are also benefits in the construction of the BEM system of equations aside from the reduced number of terms that must be updated by employing an intelligent re-meshing scheme. Since the BEM requires far fewer nodes than the FEM there are far fewer DOF; this means that the system of equations is much smaller and hence requires fewer operations to solve. However, it should be noted that the BEM system matrix is more densely populated than a FEM matrix and some of the acceleration schemes that can be applied to sparse matrices cannot, therefore, be used.

Several commercial packages that incorporate the BEM are available. These include *CA* [1] and *BEASY* [31].

1.5 Definitions

1.5.1 Real-time

Joldes *et al.* [32] apply 'real-time' techniques to neurosurgical simulation. However, in this context they define real-time to mean that the analysis should take under a minute, having established that this is the maximum amount of time a surgeon would be prepared to wait for results during surgery. This is clearly not the almost instantaneous feedback required for dynamic interaction with a model. For clarity it is therefore necessary to precisely define the terms 'real-time' and 'rapid' analysis within the context of this thesis.

The definitions of Margetts *et al.* [6] are used where 'real-time' refers to the analysis taking place within the refresh rate of the media on which the model is viewed. This will typically be within 0.05

seconds. The author has introduced the term 'rapid', which incorporates real-time but allows that the analysis may be slightly slower, taking place in an acceptably short period of time that the user is prepared to wait. This is of the order of seconds and is synonymous with the definition of 'interactive' given in [6].

1.5.2 Acceptable accuracy

Through discussion with industry partners it has been established that, for a final analysis, it is necessary that the results are within 1% accuracy. However, during dynamic updating of stress contours on continuously changing geometries, such as when a hole is dragged from one location to another, requirements on the error associated with each individual simulation can be relaxed. It is, however, desirable that the error should remain small and therefore a cap of 4% maximum error has been imposed. Once the current interactive geometric operation has been completed, a more refined mesh can be used to generate a more accurate estimation of the stress distribution.

1.5.3 Problem size

Large problems are not analysed in this work. These are considered as models with more than 10,000 DOF. The small models this research is primarily concerned with generally have fewer than 4,000 DOF. Intermediate models (4,000-10,000 DOF) are analysed in this work but are not expected to be amenable to real-time analysis at present. The maximum number of DOF required by a FE model to generate an equivalent mesh to a BE model increases approximately linearly with the size of the model. Up to 12,000 DOF are required by a FE solver to analyse a model equivalent to a BE model with 4,000 DOF. Up to 45,000 DOF are required by a FE solver to analyse a model equivalent to a BE model with 10,000 DOF. This makes the BEM a powerful technique for reducing the size of a problem.

1.5.4 General terms

Throughout this thesis, the term 're-mesh' refers to updating part of a mesh due to some geometric change whilst leaving the majority of the mesh unchanged. 'Re-generation' implies generation of an entirely new mesh over the entire model or a specific area of the model, such as a single face. The terms 're-integration' and 're-solution' respectively refer to updating the BEM system of equations and solving the resulting system. Combined, the re-integration and re-solution procedures form the 're-analysis'.

1.6 Thesis structure

1.6.1 Overview

In order to achieve the goals set out in Section 1.1, three areas of interest must be addressed:

1. Meshing and re-meshing.
2. BE integration and construction of the BE system of equations.
3. Solution of the BE system of equations.

Together these form the CA3D analysis package. The path a model follows through CA3D is shown in Figure 1.2.

This thesis is structured such that it can be divided into two broad parts covering the theory behind the techniques discussed and applied in this work, and how these techniques have been developed and implemented by the author. This structure is summarised in Table 1.1.

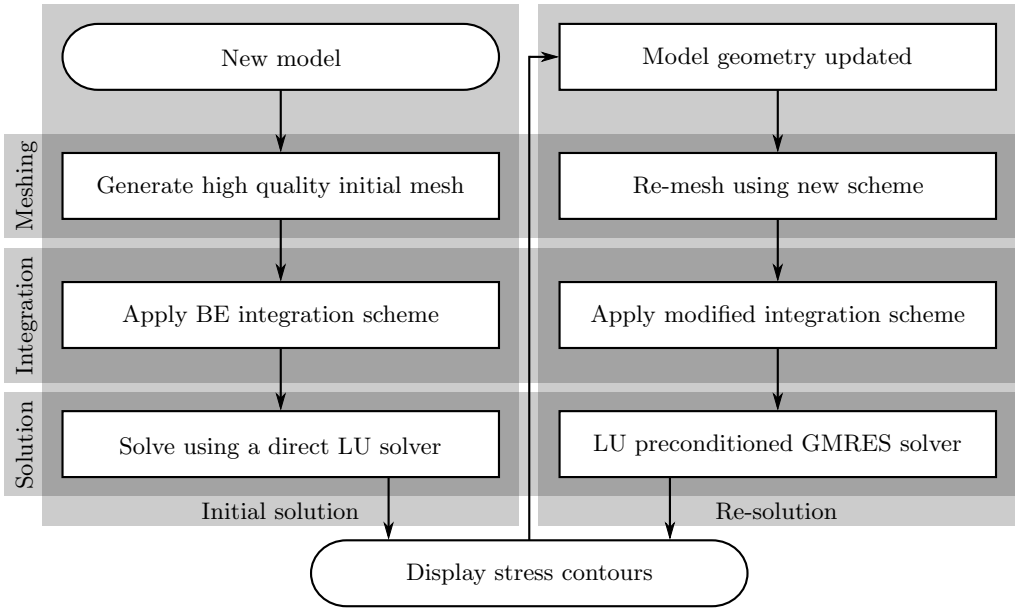


Figure 1.2: Structure of CA3D.

Table 1.1: Thesis structure.

Topic	**Theory**	**New work**
Meshing	Chapter 3	Chapter 6 Chapter 7
Integration	Chapter 2 Chapter 4	Chapter 8
Solution	Chapter 5	Chapter 9
Software implementation		Chapter 10

1.6.2 Chapter summary

This section gives a brief overview of the contents of each chapter.

Chapter 2: The boundary element method includes a history of the BEM and derives the BEM for three-dimensional elasticity. A brief review of the literature surrounding parallelisation of the BEM is also given.

Chapter 3: Mesh Generation introduces the properties of meshes and the element types typically encountered in the BEM. A comprehensive literature review covering a wide range of mesh generation techniques and data structures is given.

Chapter 4: Numerical integration covers the theory behind the numerical integration techniques used in this work. Techniques to accelerate the integration schemes are also discussed, including methods to reduce the time required to carry out a full integration, and to reduce the number of integrals required such as through adaptive cross approximation (ACA).

Chapter 5: Linear equation solvers reviews the theory behind direct, iterative and reduction based methods for solving dense linear systems. Flowcharts of the methods are also included along with suggestions for parallelisation of the techniques.

Chapter 6: Initial meshing strategy gives a guide to the data structures and initial meshing algorithms developed by the author for use in the CA3D package. Measures to determine the quality of

elements within a mesh and of the global mesh are introduced and appraised.

Chapter 7: Re-meshing to accommodate a geometrical change introduces the re-meshing algorithm developed as part of this work to limit the number of updated elements in the model and presents a comprehensive validation of the scheme, giving details on the accuracy of the expected results.

Chapter 8: Acceleration of the integration phase investigates both algorithmic and computational techniques to accelerate the computation of the boundary integral equation (BIE) which populates the BE linear system of equations. The benefits and drawbacks of applying ACA are also discussed.

Chapter 9: Acceleration of the solution phase gives details of the implementation of the direct, iterative and reduction based solvers developed for CA3D, including pseudo-codes. These solvers are compared to find the fastest and most accurate technique for solving small BE problems. Techniques to parallelise the solvers are also implemented.

Chapter 10: Evaluation combines the algorithms which have been independently implemented in the previous four chapters into a single software. This software is evaluated for real world models. An overview of the CA3D graphical user interface (GUI), which concentrates on the simplicity of the modelling environment, is also included.

Chapter 11: Conclusion summarises the work carried out for this thesis and provides detailed suggestions for how this work should be further developed and alternative applications for which it could be used.

Chapter 2

The boundary element method

"He who loves practice without theory is like the sailor who boards ship without a rudder and compass and never knows where he may cast."

Leonardo da Vinci

2.1 History of the boundary element method

The roots of the boundary element method (BEM) can be traced back to the 19th century with the foundations of the method found in the works of eminent mathematicians such as Laplace, Gauss, Fredholm, Betti, Muskhelishvili and Mikhlin. Early work concentrated on the development of the boundary integral equations (BIEs) for analytical solution of boundary value problems. Eventually, with the advent of increased computing power and availability, the method was identified as a suitable technique for computational analysis. The BEM is now used as an alternative to the finite element method (FEM). This review gives a brief history of the development of the BEM. If a complete, in depth history is required, the reader is referred to [33].

The first researchers to discretise the integral equations associated with the two-dimensional BEM were Jaswon [34] and Symm [35]. They used constant elements to solve potential problems. General integration was carried out using *Simpson's rule* whilst singular integrals were solved analytically.

There are two forms of the BEM, direct and indirect. When applying the indirect form, the solution is given in terms of the density of a series of fictitious sources acting on the boundary. Once found these sources can be used to find the actual physical parameters through a basic integration routine. The direct method uses continuous parameters applied to the boundary and is more suited to stress analysis. This thesis will therefore focus entirely on the direct method.

In 1967, Rizzo published a paper [36] that is regarded by many researchers as the beginning of a novel direct boundary integral method for the numerical solution of elasticity problems. Rizzo was the first to combine potential theory with classical elasticity theory, thereby creating a numerical approach to solving boundary value problems by directly applying displacements and tractions in a boundary integral equation. This work was continued by Cruse and Rizzo [37, 38] and extended into three dimensions by Cruse [39, 40]. Cruse initially used much the same formulation as Rizzo [36] for the three-dimensional problem and meshed the model using plane triangular elements with constant tractions and displacements. However, he later allowed more sophisticated analysis by introducing linear elements [41]. These allow tractions and displacements to vary linearly across the element.

During the early seventies, the BEM was developed to cover a wide range of applications including elastoplasticity [42], anisotropic materials [43] and fracture [44]. However, it was not until 1977 that the term BEM was coined by Brebbia and Dominguez [45]. Up until this point, most authors used the term

boundary-integral method or technique. From hereon, the BEM has developed rapidly and is now one of the major alternatives to the FEM for linear problems in all areas of engineering. Recent advances have also led to applications in non-linear elasto-plastic problems [46].

The BEM is most often used for problems that lend themselves well to the method, such as infinite domain problems and those involving discontinuities. This research aims to use the boundary only meshing to advantage in everyday mechanical elasticity calculations. The recent emergence of isogeometric analysis as a tool for analysing models directly from computer aided design (CAD) geometry, for which the BEM is also suited, is further propagating the BEM into the mechanical analysis of solids.

2.2 Derivation of the boundary integral equation

2.2.1 The problem domain

The general form of the BIE will be derived for the general two-dimensional elasticity case shown in Figure 2.1 with problem domain, Ω and boundary Γ. The method will later be extended into three dimensions. It is assumed that prescribed conditions are only applied to the boundary, although other formulations exist which can deal with body forces.

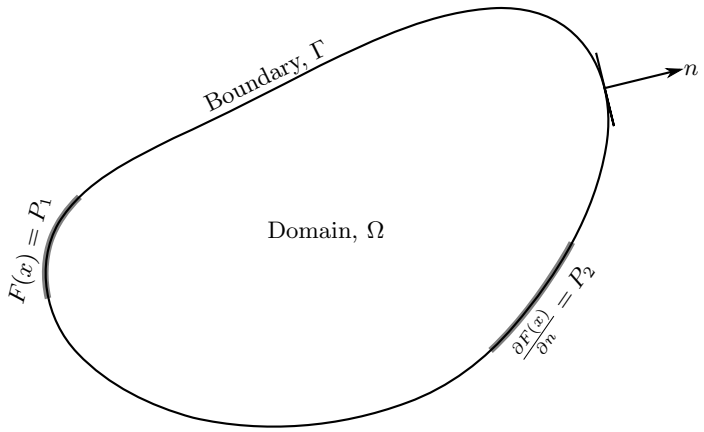


Figure 2.1: A general boundary value problem.

Normally two different forms of boundary condition may be applied to the boundary, *Dirichlet conditions*:

$$F(x) = P_1 \tag{2.1}$$

where P_1 is a known quantity and $F(x)$is an unknown function, and *Neumann conditions*:

$$\frac{\partial F(x)}{\partial n} = P_2 \tag{2.2}$$

where P_2 is a known quantity and n is the unit outward normal to Γ. Further non-linear conditions exist, such as contact problems, but these require special treatment as discussed by Aliabadi [47]

2.2.2 Green's theorum

Integral functions that link the boundary functions, $F(x)$ and $\frac{\partial F(x)}{\partial n}$, can be constructed. Integral transformations, such as *Green's Theorem*, can be applied to convert the volume integral into a surface integral:

$$\int_{\Omega} \nabla \cdot G d\Omega = \int_{\Gamma} G \cdot n d\Gamma \tag{2.3}$$

where G is a arbitrary function with continuous derivatives with respect to the Cartesian coordinate axes. Other forms of Green's theorem exist. However, as equation (2.3) is the only form encountered in this work it will be referred to as Green's theorem.

2.2.3 The reciprocal theorem

Throughout this derivation, Einstein's summation convention is used where subscripts are repeated.

The equations of equilibrium at a point in a domain Ω are given in tensor notation as:

$$\sigma_{ij,j} + b_i = 0 \tag{2.4}$$

where σ_{ij} is a stress component and b_i represents the body forces per unit volume at this point. The i and j subscripts define Cartesian directional components. The weighted residual form of the equilibrium equation can be constructed:

$$\int_\Omega (\sigma_{ij,j} + b_i) u_i^* d\Omega = \int_\Omega (\sigma_{ij,j} u_i^* + b_i u_i^*) d\Omega = 0 \tag{2.5}$$

where u_i^* is a weighting function related to the displacement field generated from some arbitrary loads. This weighted residual form of the equation of equilibrium must be expanded so that Green's theorem can be applied. Considering the product rule whereby:

$$(\sigma_{ij} u_i^*)_{,j} = \sigma_{ij,j} u_i^* + \sigma_i u_{i,j}^* \tag{2.6}$$

and applying this to equation (2.5) produces:

$$\int_\Omega \left[(\sigma_{ij} u_i^*)_{,j} - \sigma_{ij} u_i^* + b_i u_i^* \right] d\Omega = 0 \tag{2.7}$$

It should be noted that $(\sigma_{ij} u_i^*)_{,j}$ is the divergence of $\sigma_{ij} u_i^*$ and Green's theorem (equation (2.3)) can be used to transform this part of the formulation into a surface integral:

$$\int_\Gamma \sigma_{ij} u_i^* n_j d\Gamma - \int_\Omega \sigma_{ij} u_{i,j}^* d\Omega + \int_\Omega b_i u_i^* d\Omega = 0 \tag{2.8}$$

If tractions, t_i, are defined using the *Cauchy stress transformation* such that:

$$t_i = \sigma_{ij} n_j \tag{2.9}$$

the boundary integral can be simplified:

$$\int_\Gamma u_i^* t_i d\Gamma - \int_\Omega u_{i,j}^* \sigma_{ij} d\Omega + \int_\Omega u_i^* b_i d\Omega = 0 \tag{2.10}$$

thereby producing the *principle of virtual work*.

A second virtual work expression can be conceived, whereby a fictitious load case (denoted by the asterisk) does work on the real displacements, u_i:

$$\int_\Gamma t_i^* u_i d\Gamma - \int_\Omega \sigma_{ij}^* u_{i,j} d\Omega + \int_\Omega b_i^* u_i d\Omega = 0 \tag{2.11}$$

where t_i^* is a weighting function related to the traction field generated by the fictitious load case.

From *Hooke's law* it can be shown that $\sigma_{ij} u_{i,j}^* = \sigma_{ij}^* u_{i,j}$ and hence we arrive at *Betti's reciprocal*

theorem:

$$\int_\Gamma t_i^* u_i d\Gamma + \int_\Omega b_i^* u_i d\Omega = \int_\Gamma u_i^* t_i d\Gamma + \int_\Omega u_i^* b_i d\Omega \tag{2.12}$$

which allows the two load cases to be related via their respective displacement fields.

2.2.4 The fundamental solutions

It is now necessary to define the fictitious load cases applied to the model so that the integration can be expressed purely as a surface integral and the fundamental solutions established.

If it is assumed that the body forces, b, in the real problem do not exist, the volume integral on the right hand side of equation (2.12) can be eliminated. If it is assumed that the second fictitious load case takes the form of a *Dirac delta function* applied at source point, $p \in \Omega$:

$$\sigma_{ij,j} + \Delta(q-p)e_i(q) = 0 \tag{2.13}$$

where q is an arbitrary field point and e_i is a unit vector in the coordinate direction, i. The Dirac delta function has discontinuous properties:

$$\Delta(q-p) = \begin{cases} \infty & \text{if } p = q \\ 0 & \text{if } x \neq p \end{cases} \tag{2.14}$$

It should be noted that the zero to infinite discontinuity at $p = q$ means that the load case indicated by the asterisk cannot assume any physical meaning in continuum mechanics. Equation (2.12) can now be re-written:

$$\int_\Gamma t_i^* u_i d\Gamma + \int_\Omega \Delta(q-p)e_i u_i d\Omega = \int_\Gamma u_i^* t_i d\Gamma \tag{2.15}$$

and, as the integral of the Dirac delta function is unity, it can be stated that:

$$\int_\Omega \Delta(q-p)e_i u_i d\Omega = e_i u_i(p) \tag{2.16}$$

thus:

$$u_i(p)e_i + \int_\Gamma t_i^* u_i d\Gamma = \int_\Gamma u_i^* t_i d\Gamma \tag{2.17}$$

If the body force vector is assumed to be the Dirac delta function, U_{ij} is given by the *Kelvin solution* to equation (2.13). For the three-dimensional problem, the displacement fundamental solution, U_{ij}, is given as:

$$U_{ij} = \frac{1}{16\pi\mu(1-\nu)r}\left[(3-4\nu)\delta_{ij} + r_{,i}r_{,j}\right] \tag{2.18}$$

where $r = |q-p|$, μ and ν are respectively the shear modulus and Poisson's ratio of the material and δ_{ij} is the *Kronecker delta*:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{2.19}$$

The displacements U_{ij} are called the fundamental solutions and represent the displacement, at field point q and in the direction i, resulting from a Dirac delta function force at point p applied in direction j. They are related to u_i^* in (2.12) by:

$$u_i^* = U_{ij}e_j \tag{2.20}$$

The three-dimensional traction fundamental solution, T_{ij}, can be found by differentiating equation

(2.18) and applying Hooke's law:

$$T_{ij} = \frac{-1}{8\pi(1-\nu)r^2} r_{,n} \left[(1-2\nu)\delta_{ij} + 3r_{,i}r_{,j}\right] + \frac{1-2\nu}{8\pi(1-\nu)r^2}(r_{,j}n_i - r_{,j}) \tag{2.21}$$

The relationship between T_{ij} and t_i^* is similar to the corresponding displacement relationship (2.20):

$$t_i^* = T_{ij}e_j \tag{2.22}$$

2.2.5 The boundary integral equation

Incorporating the fundamental solutions, equation (2.17) can now be written:

$$u_i(p) + \int_\Gamma T_{ij}u_j d\Gamma = \int_\Gamma U_{ij}t_j d\Gamma \tag{2.23}$$

Equation (2.23) now only includes one term that is not dependent on the boundary, $u_i(p)$. As the location of p is still arbitrary, this can be remedied simply by enforcing the condition, $p \in \Gamma$. However, this introduces complications in the integration of equation (2.23) as the fundamental solutions U_{ij} and T_{ij} are functions of $1/r$ and $1/r^2$, rendering them weakly and strongly singular respectively. As $r = \|p - q\|$, these functions will be singular when $p = q$.

The weakly singular U_{ij} kernel does not generally present a problem. However, the T_{ij} kernel must be integrated in the *Cauchy principal value* sense. This defines the integral as the limiting value of the integral over the entire boundary, except the point at which the singularity occurs where the integral is considered over a definition of Γ that has been split into $\Gamma - \Gamma_\varepsilon$ and Γ_ε^+, as shown in Figure 2.2.

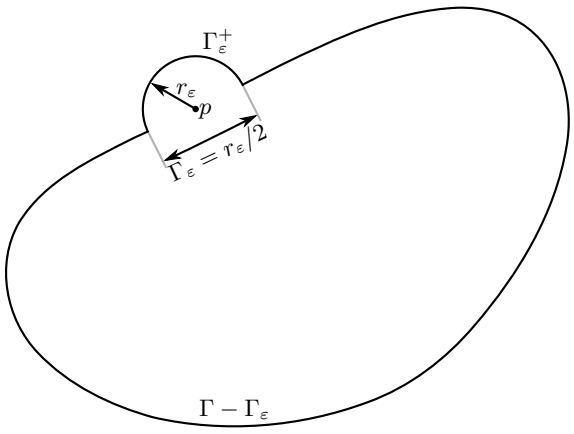


Figure 2.2: Integration in the Cauchy principal value sense.

In two dimensions, Γ_ε^+ forms a semicircle surrounding p. In three dimensions, Γ_ε^+ represents a hemisphere. The integral is now re-written in three parts considering the limit as $r_\varepsilon \to 0$:

$$\int_\Gamma T_{ij}u_j d\Gamma = \lim_{r_\varepsilon \to 0}\left(\int_{\Gamma-\Gamma_\varepsilon} T_{ij}u_j d\Gamma\right) + \lim_{r_\varepsilon \to 0}\left(\int_{\Gamma_\varepsilon^+} T_{ij}(u_j(q) - u_j(p))d\Gamma(q)\right) + u_j(p)\lim_{r_\varepsilon \to 0}\left(\int_{\Gamma_\varepsilon^+} T_{ij}d\Gamma\right) \tag{2.24}$$

where the first term on the right hand side is the Cauchy principal value integral. The second term will vanish since as $r_\varepsilon \to 0$, $u_j(q) \to u_j(p)$. The final term can be simplified:

$$u_j(p)\lim_{r_\varepsilon \to 0}\left(\int_{\Gamma_\varepsilon^+} T_{ij}d\Gamma\right) = \alpha_{ij}(p)u_j(p) \tag{2.25}$$

By substituting the now simplified version of equation (2.24) into equation (2.23), the BIE is formed:

$$c_{ij}(p)u_i(p) + \int_\Gamma T_{ij}u_j d\Gamma = \int_\Gamma U_{ij}t_j d\Gamma \tag{2.26}$$

where, for a smooth boundary:

$$c_{ij}(p) = \delta_{ij} + \alpha_{ij}(p) = \frac{\delta_{ij}}{2} \tag{2.27}$$

and, for angular boundaries in the three-dimensional case:

$$c_{ij}(p) = \frac{\beta}{4\pi} \tag{2.28}$$

where β is the angle subtended by the material within the boundary at p.

2.3 Discretisation of the boundary integral equation

To solve the system numerically, the boundary, Γ, must first be discretised into a series of elements, Γ_e ($e = 1, 2, 3, \ldots, n_E$), which form a surface mesh. Methods to generate this mesh will be discussed in Section 3. The discretised BIE can be re-written:

$$c_{ij}(p)u_i(p) + \sum_{e=0}^{n_E} \int_{\Gamma_e} T_{ij}u_i d\Gamma = \sum_{e=0}^{n_E} \int_{\Gamma_e} U_{ij}t_i d\Gamma \tag{2.29}$$

If the u_i and t_i terms are expressed in their interpolated forms, defined by shape functions, N_k, they cease to be functions of position along the boundary and can therefore be removed from the integrals:

$$c_{ij}(p)u_i(p) + \sum_{e=0}^{n_E} \int_{\Gamma_e} T_{ij}N_k d\Gamma \; u_i^{ke} = \sum_{e=0}^{n_E} \int_{\Gamma_e} U_{ij}N_k d\Gamma \; t_i^{ke} \tag{2.30}$$

where t_i^{ke} and u_i^{ke} are the tractions and displacements acting in the i direction at node k of element e. The shape function N_k is the value of the interpolation function for node k at the current integration point. These shape functions are discussed further in Section 4.2.

In order to perform numerical integration, the elements, Γ_e, are transformed into the local parametric coordinate system, (ξ, η):

$$c_{ij}(p)u_i(p) + \sum_{e=0}^{n_E} \int_{-1}^{1}\int_{-1}^{1} T_{ij}N_k J d\xi d\eta \; u_i^{ke} = \sum_{e=0}^{n_E} \int_{-1}^{1}\int_{-1}^{1} U_{ij}N_k J d\xi d\eta \; t_i^{ke} \tag{2.31}$$

where J is the Jacobian, which transforms the differential variables at the current point from the local into the global coordinate systems, ie. $d\Gamma = J(\xi, \eta)d\xi d\eta$.

2.4 Boundary element method matrix equations

2.4.1 Matrix assembly

The integrals contained within equation (2.31) can be computed using a Gauss-Legendre quadrature by placing p at each node in turn and q at integration points spread across each element. The precise integration scheme will be discussed further in Section 4.2. The integrals are combined to form a system of equations which is given in matrix form as:

$$[H]\{u\} = [G]\{t\} \tag{2.32}$$

where $[H]$ and $[G]$ contain the integrated traction and displacement kernels respectively. Matrix $[H]$ is square and $[G]$ is rectangular to allow for discontinuous tractions.

It should be noted that the terms along the diagonal of $[H]$ are more complex to compute than the other terms in the matrix due to the strongly singular traction kernel, T_{ij}. Whilst these terms could be computed directly [48], this is not always necessary and simpler techniques can be applied to the majority of problems. In this work they are constructed using the row-sum technique. In three dimensions these terms form 3×3 blocks along the diagonal and can be calculated by carrying out a unit translation of the entire model along each coordinate axis in turn. For the solution in the x direction, $\{t\} = 0$ for all entries and $\{u\} = \{1, 0, 0, 1, 0, 0, \ldots\}$. The first column of each block can now be computed as the negative of the sum of every third term in the associated row of $[H]$. The same technique can be applied in the y and z directions. However, care should be taken as the diagonal terms dominate the system and will accumulate any errors in the off diagonal terms when the row-sum is applied. This computationally inexpensive technique is called *rigid body motion*.

2.4.2 The linear system

For the two-dimensional case, the system contains $n = 2n_N$ equations, where n_N is the number of nodes. In three dimensions, $n = 3n_N$. This matrix system given in equation (2.32) contains n equations but $2n$ unknowns, n values must therefore be provided as boundary conditions to make solution possible. As the majority of nodes, q, usually lie on a free surface, $t(q) = 0$. The rest of the boundary conditions are obtained through tractions and displacements applied directly to q. Once an appropriate set of boundary conditions have been defined, column swapping can be applied between $[H]$ and $[G]$ and equation (2.32) rewritten in the form:

$$[A]\{x\} = [B]\{y\} \tag{2.33}$$

where $\{x\}$ now contains all the unknown tractions and displacements and $\{y\}$ contains the prescribed boundary conditions. Since both $[B]$ and $\{y\}$ are known these can be multiplied to yield:

$$[A]\{x\} = \{b\} \tag{2.34}$$

This linear system can now be solved as a set of linear equations to find the unknown tractions and displacements contained in $\{x\}$.

2.4.3 Scaling

As $\{u\}$ and $\{t\}$ often differ by several orders of magnitude, the solution is likely to suffer from poor conditioning. This problem is commonly avoided by introducing a scaling factor, ψ, by re-writing equation (2.32):

$$[H]\{u\} = \psi[G]\{\hat{t}\} \tag{2.35}$$

where:

$$\{\hat{t}\} = \psi^{-1}\{t\} \tag{2.36}$$

Applying ψ ensures that $\{u\}$ and $\{\hat{t}\}$ are of similar magnitude and leads to a more accurate solution. The precise value of ψ is not important but an appropriate magnitude should be selected. This can be determined from the maximum dimensions and the material factors of the model being analysed. The scaling will need to be reversed during stress recovery.

2.5 Stress recovery

Some work is needed to extract the boundary stresses from the solution vector, $\{x\}$, which now contains the unknown values of $\{u\}$ and $\{\hat{t}\}$. In order to extract the stresses, the local orthogonal unit axis system, (b_1, b_2, b_3), which is tangential to Γ, must be defined as shown in Figure 2.3. Vectors a_1 and a_2, which are tangential to ξ and η respectively and separated by angle ϑ, must also be defined.

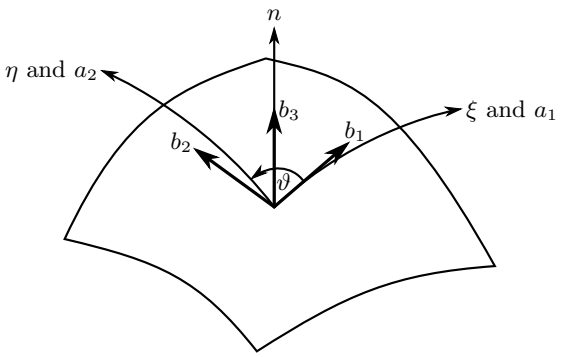


Figure 2.3: Local axes used in stress recovery.

The strains, ε in directions tangential to the three dimensional boundary, Γ, may be computed from the following equations:

$$\varepsilon_{11} = \frac{1}{|a_1|}\left[\frac{\partial u}{\partial \xi}\cdot b_1\right] \tag{2.37}$$

$$\varepsilon_{22} = \frac{-\cos\vartheta}{|a_1|\sin\vartheta}\left(\frac{\partial u}{\partial \xi}\cdot b_2\right) + \frac{1}{|a_2|\sin\vartheta}\left[\frac{\partial u}{\partial \eta}\cdot b_2\right] \tag{2.38}$$

$$\varepsilon_{12} = \frac{-\cos\vartheta}{|a_1|\sin\vartheta}\left(\frac{\partial u}{\partial \xi}\cdot b_1\right) + \frac{1}{|a_2|\sin\vartheta}\left[\frac{\partial u}{\partial \eta}\cdot b_1\right] + \frac{1}{|a_1|}\frac{\partial u}{\partial \xi}\cdot b_2 \tag{2.39}$$

where

$$\frac{\partial u}{\partial \xi} = \begin{Bmatrix} \frac{\partial u_x}{\partial \xi} \\ \frac{\partial u_y}{\partial \xi} \\ \frac{\partial u_z}{\partial \xi} \end{Bmatrix} \qquad \frac{\partial u}{\partial \eta} = \begin{Bmatrix} \frac{\partial u_x}{\partial \eta} \\ \frac{\partial u_y}{\partial \eta} \\ \frac{\partial u_z}{\partial \eta} \end{Bmatrix} \tag{2.40}$$

$|a_1|$ and $|a_2|$ are the magnitudes of the tangential vectors to ξ and η, given by:

$$|a_1| = \sqrt{\left(\frac{\partial x}{\partial \xi}\right)^2 + \left(\frac{\partial y}{\partial \xi}\right)^2 + \left(\frac{\partial z}{\partial \xi}\right)^2} \tag{2.41}$$

$$|a_2| = \sqrt{\left(\frac{\partial x}{\partial \eta}\right)^2 + \left(\frac{\partial y}{\partial \eta}\right)^2 + \left(\frac{\partial z}{\partial \eta}\right)^2} \tag{2.42}$$

The local tangential stresses can be found by substituting ε and the local orthogonal tractions, t_i, into Hooke's law:

$$\sigma_{11} = \frac{E}{1-2\nu^2}(\varepsilon_{11} + \nu\varepsilon_{22}) + \frac{\nu}{1-\nu}t_3 \tag{2.43}$$

$$\sigma_{22} = \frac{E}{1-2\nu^2}(\varepsilon_{22} + \nu\varepsilon_{11}) + \frac{\nu}{1-\nu}t_3 \tag{2.44}$$

$$\sigma_{12} = \frac{E}{2(1+\nu)}\varepsilon_{12} \tag{2.45}$$

$$\tag{2.46}$$

$$\sigma_{33} = t_3 \tag{2.47}$$
$$\sigma_{13} = t_2 \tag{2.48}$$
$$\sigma_{23} = t_1 \tag{2.49}$$

where t_i is extracted from the scaled values contained in the solution vector, $\{x\}$:

$$t_1 = \psi(\hat{t} \cdot b_1) \tag{2.50}$$
$$t_2 = \psi(\hat{t} \cdot b_2) \tag{2.51}$$
$$t_3 = \psi(\hat{t} \cdot n) \tag{2.52}$$

The global stresses may now be found by applying a transformation matrix:

$$\begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{zx} \end{bmatrix} = \begin{bmatrix} b_{3x}^2 & b_{1x}^2 & b_{2x}^2 & 2b_{3x}b_{1x} & 2b_{1x}b_{2x} & 2b_{3x}b_{2x} \\ b_{3y}^2 & b_{1y}^2 & b_{2y}^2 & 2b_{3y}b_{1y} & 2b_{1y}b_{2y} & 2b_{3y}b_{2y} \\ b_{3z}^2 & b_{1z}^2 & b_{2z}^2 & 2b_{3z}b_{1z} & 2b_{1z}b_{2z} & 2b_{3y}b_{2z} \\ b_{3x}b_{3y} & b_{1x}b_{1y} & b_{2x}b_{2y} & b_{3x}b_{1y} + b_{3y}b_{1x} & b_{1x}b_{2y} + b_{1y}b_{2x} & b_{3x}b_{2y} + b_{3y}b_{2x} \\ b_{3y}b_{3z} & b_{1y}b_{1z} & b_{2y}b_{2z} & b_{3y}b_{1z} + b_{3z}b_{1y} & b_{1y}b_{2z} + b_{1z}b_{2y} & b_{3y}b_{2z} + b_{3z}b_{2y} \\ b_{3z}b_{3x} & b_{1z}b_{1x} & b_{2z}b_{2x} & b_{3z}b_{1x} + b_{3x}b_{1z} & b_{1z}b_{2x} + b_{1x}b_{2z} & b_{3z}b_{2x} + b_{3x}b_{2z} \end{bmatrix} \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{12} \\ \sigma_{23} \\ \sigma_{13} \end{bmatrix} \tag{2.53}$$

2.6 Internal solution

It is often necessary to calculate the stress, σ, and displacement, u, across the interior of a domain. The internal displacement, u can be found by placing source point p at the point of interest, substituting the now fully defined values of displacement and traction on Γ into equation (2.31) and summing boundary integrals to yield $u(p)$.

The internal stress components may be derived from a derivative of the BIE:

$$\sigma_{ij}(p) + \int_\Gamma S_{kij} u_k d\Gamma = \int_\Gamma D_{kij} t_k d\Gamma \tag{2.54}$$

where the kernels, S_{kij} and D_{kij} can be found by differentiating T_{ij} and U_{ij} respectively. For the three-dimensional case:

$$\begin{aligned} S_{kij} &= \frac{\mu}{4\pi(1-\nu)r^3} n_i \left[3\nu r_{,j} r_{,k} + (1-2\nu)\delta_{jk}\right] \\ &+ \frac{\mu}{4\pi(1-\nu)r^3} n_j \left[3\nu r_{,i} r_{,k} + (1-2\nu)\delta_{ik}\right] \\ &+ \frac{\mu}{4\pi(1-\nu)r^3} n_k \left[3(1-2\nu) r_{,i} r_{,j} - (1-4\nu)\delta_{ij}\right] \\ &+ \frac{3\mu}{4\pi(1-\nu)r^3} r_{,n} \left[(1-2\nu)\delta_{ij} r_{,k} + \nu\left(\delta_{jk} r_{,i} + \delta_{ik} r_{,j}\right) - 5 r_{,i} r_{,j} r_{,k}\right] \end{aligned} \tag{2.55}$$

$$D_{kij} = \frac{1}{8\pi(1-\nu)r^2} \left[(1-2\nu)\left(\delta_{jk} r_{,i} + \delta_{ik} r_{,j} + \delta_{ij} r_{,k}\right) + 3 r_{,i} r_{,j} r_{,k}\right] \tag{2.56}$$

It should be noted that, for the internal problem, $p \notin \Gamma$ and there are therefore no singular integrals in the system. However, care must be taken if p is close to Γ as S_{kij} and D_{kij} exhibit a higher order of singularity than T_{ij} and U_{ij}.

2.7 Parallelisation of the boundary element method

The first algorithm for parallelisation of the BEM appeared in 1984 in a work by Symm [49]. Davies [50] comprehensively reviews the early work on parallelising the method. Many authors, including Kane [51] and Baltz and Ingber [52], use block partitioning to distribute different parts of the $[A]$ matrix over multiple processors. However the overlap between these blocks, where elements share a node, adds an extra level of complexity. Erhart [53] and Kamiya *et al.* [54, 55] use domain decomposition to solve the boundary element (BE) problem. This is easily parallelised as each domain can be allocated to a different processor. However, additional overheads are incurred in assessing the performance of each processor for optimal distribution of the computation.

Kreienmeyer and Stein [56] show very good speed-up and scalability when assembling the BEM system of equations on multiple central processing units (CPUs), typically reporting 95% efficiency. They also compare the speed-up achieved by parallelising several different solver routines.

Cunha *et al.* [57] accelerate the BEM by formulating it for clusters and supercomputers. They do not intend this technique to provide interactivity, only to accelerate solution. Stock and Gharakhani [58] apply the BEM to cortex particle methods to solve systems with up to 1.4 million unknowns using hybrid CPU/graphics processing unit (GPU) parallelisation. A 20-fold speed-up is reported in generating the solution. The majority of this speed gain is made during calculation of the discretised BIE given in Section 2. This speed up is in line with that achieved for finite element (FE) analysis on the GPU by Joldes *et al.* [32], enabling analysis of FE models with up to 50,000 degrees of freedom (DOF) in under a minute.

More recent research has focussed on parallelising the BEM for GPUs. A good introduction to GPUs can be found in [59].

Chapter 3

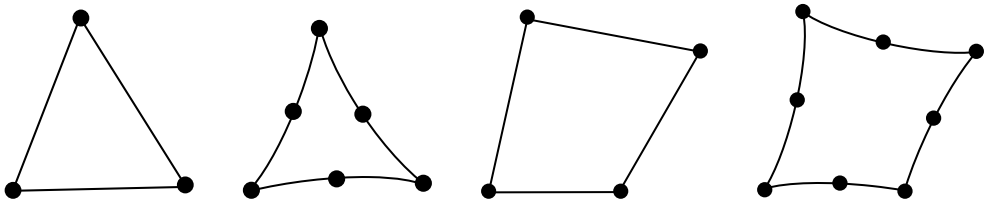
Mesh generation

"Employ your time in improving yourself by other men's writings so that you shall come easily by what others have laboured hard for."

Attributed to Socrates

3.1 Introduction

The various meshing schemes available to the engineer are compared in this chapter. These are used to discretise the surface, or boundary, of a model so that the boundary element method (BEM) can be implemented. In three-dimensional boundary element (BE) analysis, these elements usually take the form of two-dimensional quadrilaterals or triangles as illustrated in Figure 3.1. These two-dimensional elements may also be mapped onto a three-dimensional surface [60]. It is worth noting the similarity between three-dimensional BEs and two-dimensional finite elements (FEs). Comparing a single face of a three-dimensional BE model to a two-dimensional FE model, it can be seen that both methods require that a series of similar elements be generated over the domain.



(a) Linear triangle. (b) Quadratic triangle. (c) Linear quadrilateral. (d) Quadratic quadrilateral.

Figure 3.1: Element types.

This review encompasses two-dimensional FE meshing and its applicability as a method for three-dimensional surface meshing as well as specialised boundary meshing schemes. Different element types will be discussed and the drawbacks and benefits they can bring to the BEM will be evaluated. A variety of FE mesh generation schemes will be compared and their effectiveness assessed. A brief history of the development of automated meshing can be found in [61]. Topping *et al.* also discuss several of the primary meshing procedures along with suggested further reading. Baker [62] provides a comprehensive summary of the mesh generation techniques available, with particular reference to applications in aerospace, over the past few decades.

3.2 Properties of a good mesh

Many authors provide rules detailing good practice in mesh generation and what is required of an automatic mesh generation algorithm. Canaan *et al.* [63] summarise these by outlining six key points to be considered when designing an automatic mesh generator:

- Basic functionality
- Robustness/reliability/dependability
- Mesh quality
- Speed
- Minimal required user interaction (the automatic meshing algorithm must be 'model aware'; recognising and adapting the mesh to allow for any key geometry or loading conditions)
- Controllability (how easy it is to guide mesh refinement)

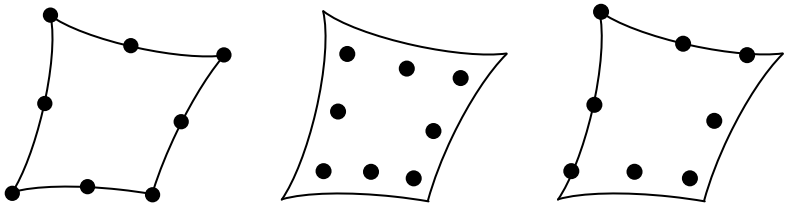
Other authors include memory requirements in this list. A compromise often has to be reached between speed and storage. However, modern computing technology means an ever-increasing amount of storage is becoming available and the current work can therefore concentrate primarily on speed.

3.3 Elements

The various linear and quadratic element types available to the engineer are illustrated by Trevelyan [30]. There are two main families of elements, continuous and discontinuous. Continuous elements share nodes, located around the boundary of the element, with neighbouring elements. The nodes on discontinuous elements are located just inside the element as shown in Figure 3.2 and are not shared with neighbouring elements. Discontinuous meshing allows elements to be generated in such a way that the corners of the elements do not match up with neighbouring elements. The two mesh types are compared in Figure 3.3. Discontinuous elements are not commonly used in the finite element method (FEM) as they introduce awkward constraint equations that are required to tie the mesh together. This does not occur in the BEM where discontinuous elements can enable more flexibility in the mesh generation scheme. Trevelyan [30] outlines the advantages and disadvantages of using these types of elements and concludes that discontinuous elements should primarily be used where there are stress discontinuities in the model. They can also be used to remove the need for nodes to match between different zones in the model. It can be argued that continuous elements share nodes and therefore have increased performance due to the shorter run time required for the reduced number of nodes in the model; however, fewer discontinuous elements are required to mesh the same region with the same numerical accuracy because of the greater number of nodes. This suggests that both element types may perform with a similar efficiency. Most authors favour continuous elements but see uses for discontinuity in some circumstances. When both element types are used to model a structure, hybrid elements are required to make the transition. All three element continuity types are illustrated in Figure 3.2.

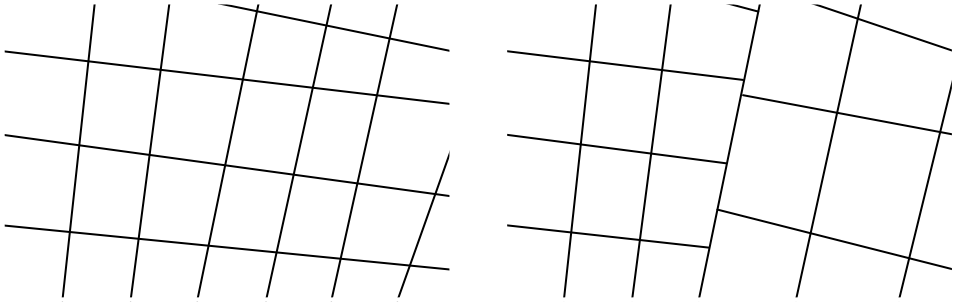
The elements used in the three-dimensional BEM are either three or four-sided and have varying order, dictated by the number of nodes located along each edge. For two-dimensional FE analysis, it is suggested that quadratic quadrilateral elements provide the most efficient and accurate analysis when compared with linear, quadratic and cubic triangular and quadrilateral elements [64]. This may not necessarily be applicable to the BEM and merits further research. The reduced number of nodes required by quadrilaterals over triangles to model the same area and the ease with which they may be interpolated over some surfaces, such as four-sided faces, may make them more efficient in some circumstances.

Different node layouts can be used to define identically shaped elements. Serendipity elements feature nodes only on the boundaries; Lagrange elements also contain internal nodes as shown in Figure 3.4(b).



(a) Continuous element. (b) Discontinuous element. (c) Hybrid element.

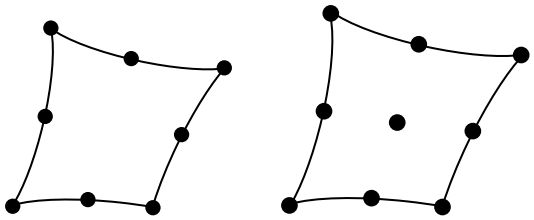
Figure 3.2: Element continuity.



(a) Continuous mesh. (b) Discontinuous mesh.

Figure 3.3: Mesh continuity.

Each different element type is defined by a different set of shape functions that are used for interpolation across the elements. Many publications list the functions used for their specific elements. Kane [29] details how to derive these functions either by inspection or algebraically. Shape functions will be discussed further in Chapter 4.



(a) Serendipity element. (b) Lagrange element.

Figure 3.4: Element types.

Element quality is briefly discussed by many authors [65–67]; the main strategies are summarised by Topping *et al.* [61]. These authors introduce formulae to assess the quality of quadrilateral or triangular elements based on the unit vectors of their edges. Other quality measures are also introduced for triangular elements. These are discussed further in Section 6.5. For quadrilateral elements, the skew and the taper, which respectively measure angular and geometric distortion from a rectangular element, can be tested along with the sizes of the internal angles.

Regular, undistorted elements generally produce smaller errors in the analysis than more heavily distorted elements [30]. Several factors that are affected by the geometry can be identified that affect the accuracy of the results:

- Irregularly spaced integration points may not capture enough detail in certain areas.

- The shape functions used to interpolate across the element may not capture the solution.
- Zero degree internal corners can create spurious peak stresses.

Poor quality elements in the correct orientation can provide accurate results. However, it is not always possible to predict what this orientation should be. The initial elements also need to be of high enough quality to accommodate a reasonable amount of distortion as the model geometry is modified before the changes propagate into surrounding elements.

It should be noted that any element, no matter how high order, will introduce errors around curved edges unless the edge is defined by a function of equal or lower order than the element. Trevelyan demonstrates the inaccuracies of modelling an arc by using three two-dimensional quadratic BEs to model a circle [30]. A sufficient number of elements must be used around any curved edge to provide a reasonable approximation.

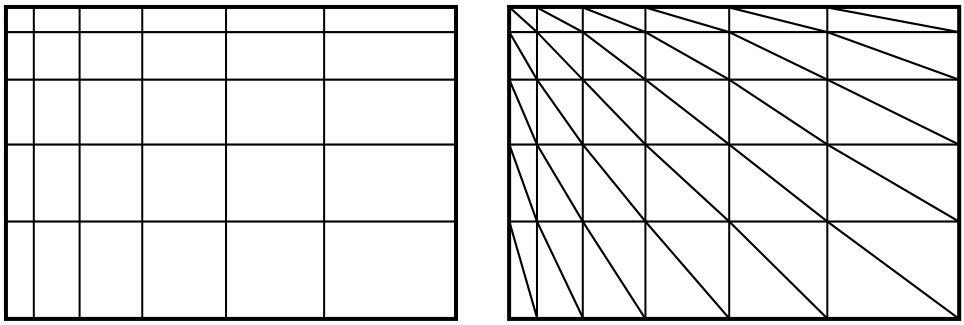
3.4 Meshing techniques

There are many different meshing techniques to be considered. These fall into two broad categories, structured and unstructured. Structured meshes generally involve constructing a grid over the model space and then refining the mesh over areas of complex geometry or where high stress gradients have been found or are expected. Unstructured meshes use a more organic approach, gradually filling in the model space with more and more elements until a required level of discretisation is reached.

Ho-Le [68] presents an early classification of the different meshing strategies which covers the majority of techniques discussed in this section.

3.4.1 Structured mesh generators

Structured meshes can be formed using several different techniques. A basic mesh can be interpolated across a surface by dividing its boundary into a series of segments and connecting them across the surface to form a grid [61, 69]. These can be further split into triangular elements, if required, as shown in Figure 3.5. This meshing technique is trivial for simple three and four-sided surfaces.

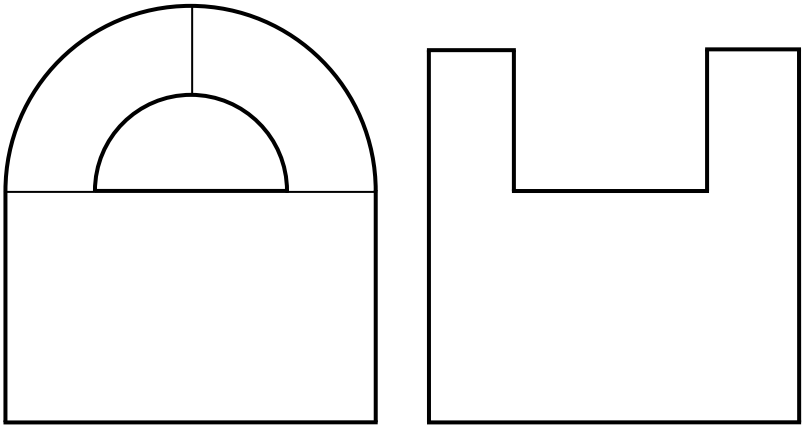


(a) Quadrilateral structured mesh. (b) Triangular structured mesh.

Figure 3.5: Basic non-uniform structured meshes.

Surfaces that are more complex require a more advanced routine. Early algorithms required the programmer to divide the surface manually into a series of four-sided patches before they could be meshed [70], as shown on Figure 3.6(a); attempts were later made to automate this process. Tam and Armstrong [71] developed a method based on medial axis subdivision. This used 'the locus of the centre of an inscribed disc of maximal diameter as it rolls around the region interior' as a guide for creating primitive sub-regions that could easily be divided into quadrilateral patches. Sub-mapping schemes can be used to convert complex geometry into a simpler shape for meshing, as is shown in Figure 3.6(b). The

mesh is then mapped back onto the original model. This simplifies the meshing procedure and ensures continuity of elements across patch boundaries in multiply-connected surfaces [72, 73]. When using BEs this is less crucial but will lead to speed benefits in the analysis.



(a) Sub-division. (b) Sub-mapping.

Figure 3.6: Sub-division and sub-mapping.

The vertices around the boundary of a model are often classified as corners, ends or sides to enable appropriate sub-mapping. However, defining this classification is not always as straight forward as it might appear. Ruiz-Gironés and Sarrate [72] introduced an advanced sub-mapping method that automates the entire structured meshing process and has the ability to reclassify commonly misclassified vertices around the model to enable mapping of more complex geometries. Subramanian *et al.* [73] manually divide a surface into quadrilateral patches, for convenience, before demonstrating their sub-mapping routine. This transforms the patches into a series of interconnected squares in the transformed plane where the user must specify the number of elements required across each row and column depending on how fine a mesh is required. The nodal coordinates are generated and serendipity shape functions used to transform back onto the model plane. This method appears to require a large amount of user input. However, techniques to divide the model automatically into suitable patches are already in existence such as the medial axis sub-division method discussed above [70]. The geometric recognition code, required to automatically specify the distribution of elements required, is also trivial to generate. Taniguchi [74] introduces a similar blocking method designed for efficiency on a microcomputer that generates a regular grid over manually specified patches. The grid is refined to match across patch boundaries.

The quadtree subdivision method lays a regular grid over the entire model space and successively refines each square until sufficient refinement has been completed to accurately represent the model [15, 75]. Figure 3.7 gives an example of this type of mesh. Rules are set which state that no element can be more than twice the size of a neighbouring element, thus the refinement propagates over the entire surface. The square elements are often divided further into triangular elements. Quadtree meshes are stored in a tree structure [15, 75, 76]. This minimises the memory required to store the mesh and aids efficiency when refining the mesh since all the data are stored in a hierarchical system. Many different schemes have been proposed to aid efficient traversal of the tree structure [15, 76, 77]. The quadtree mesh is highly adaptable and good for re-meshing; it is often used to model fluid flow where constant updating of the mesh is required [15].

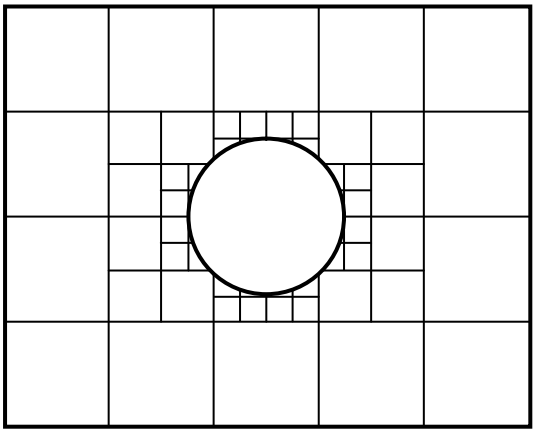
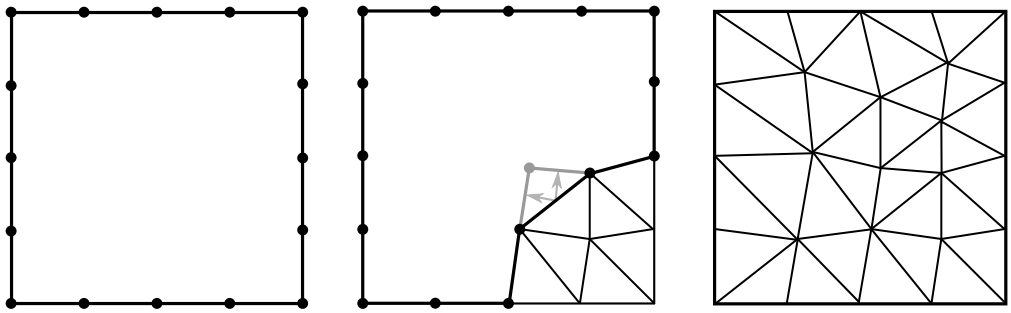


Figure 3.7: Quadtree subdivision of a plate with a hole.

3.4.2 Unstructured mesh generators

In most engineering applications unstructured mesh generators have superseded structured mesh generators as they incorporate more flexibility for meshing irregular domains. Whilst some unstructured mesh generation algorithms will produce quadrilateral elements, most produce a triangular mesh. There are two widely used methods of producing a triangular unstructured mesh, the *advancing front* method and *Delaunay triangulation*. A wide range of less common techniques have also been developed by authors for their own research.

The advancing front method was first developed by Lo in 1985 [78] and has since been adopted by many authors as one of the primary mesh generation schemes. The boundary around a surface is first split into a series of segments that make up the initial front; each segment will form a side of an element. Triangles are added to the front using segments as baselines. After each addition, the front is updated to pass around the new element with the unpopulated area contained within it. The front advances until the entire surface has been meshed. These steps are shown in Figure 3.8. For robustness, the algorithm should start with the shortest segment. Lo later went on to write an algorithm to grade the mesh based on the initial boundary segments that dispensed with the need to search for the shortest segment, thus speeding up the process and generating a better mesh gradation [79]. Many authors have developed and adapted the advancing front technique to improve its reliability, robustness and efficiency [80–82]. Topping *et al.* [61] give a good overview and history of the method.

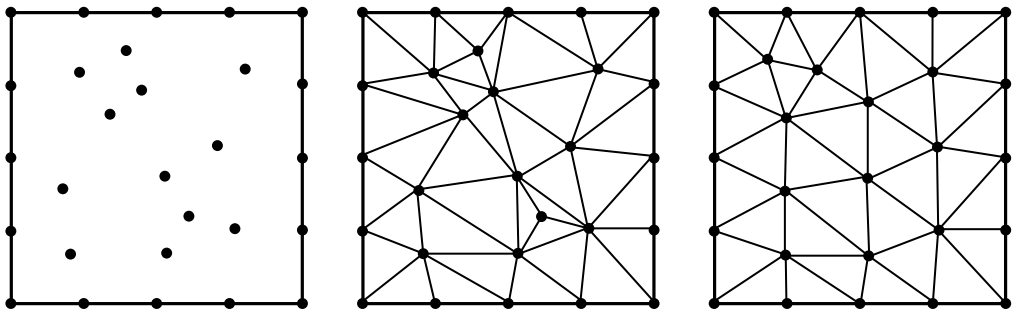


(a) Initial front. (b) Advancing the front. (c) Final mesh.

Figure 3.8: Advancing front.

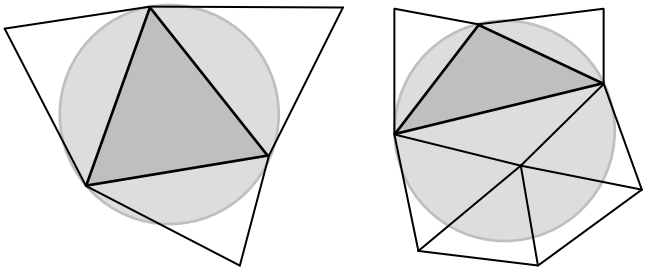
Writing in 2004, El-Hamalawi [83] describes Delaunay triangulation as 'the most efficient [unstructured] meshing technique available today in terms of speed'. Delaunay triangulation is a highly robust

and efficient method for creating a two-dimensional, triangular mesh. For a constrained triangulation, nodes are first generated around the boundaries and interior of the domain. An algorithm finds the best triangulation of the points and outputs the initial mesh. Successive stages smooth the mesh and adjust any poor quality elements by subdivision, reshaping or removal of nodes. These steps are shown in Figure 3.9. To be classified as a Delaunay triangulation no nodes should appear within the circumcircle (a circle that touches every corner of the element) of each element in the mesh, as illustrated in Figure 3.10. Much research has been done into the Delaunay method and many authors have suggested their own versions of the algorithm [61, 67, 84, 85]. The author recommends the algorithm developed by Secchi and Simoni [86] as it is particularly fast and highly robust.



(a) Initial domain. (b) Initial triangulation. (c) Smoothed mesh.

Figure 3.9: Delaunay triangulation.



(a) Delaunay triangle. (b) Non-Delaunay triangle.

Figure 3.10: Delaunay circumcircle.

Other methods have also been proposed to create a triangular mesh such as circle packing [87], shown in Figure 3.11, and the three-dimensional surface equivalent, bubble meshing [88]. These use circles and spheres to mimic the effect of Voronoi polygons, which were used as an intermediate step towards generating a Delaunay mesh before more efficient methods were developed. These techniques have not been as extensively researched as the advancing front and Delaunay methods. They do not exist in such a highly refined state and could be viewed as less reliable. However, during this research only relatively basic models will be considered and the authors have shown circle packing and bubble meshing to be reliable for simple surfaces.

Many authors have proposed methods for generating unstructured quadrilateral meshes. These range from combining and smoothing triangular meshes [86, 89] to direct generation of quadrilaterals [65, 90]. All of these authors propose that quadrilateral elements provide improved performance over triangular elements and therefore the additional computational time required to combine triangular meshes into quadrilaterals is justified. Lee and Lo [89] define a basic method that eliminates all triangular elements

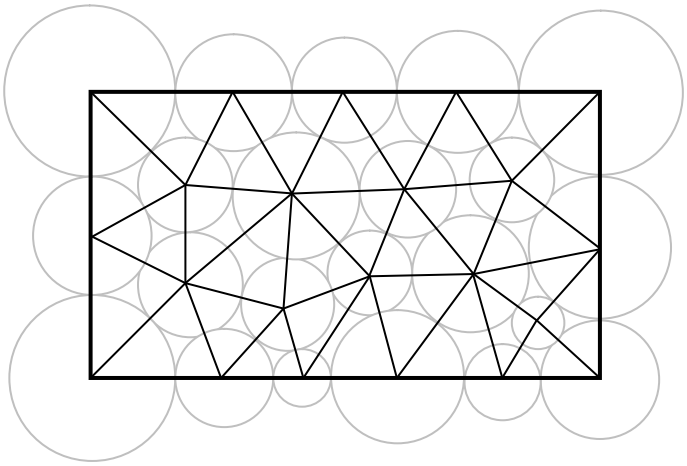


Figure 3.11: Circle packing.

from a mesh generated using the advancing front technique. This requires that an even number of initial segments be used to generate the triangular mesh. Secchi and Simoni [86] use a Delaunay triangulation to generate their initial mesh. Some triangular elements are retained within the mesh to simplify the merging procedure and to provide a better mesh around detailed geometry. Detailed information about the speed of the procedure is also provided.

Meshing techniques that can be used to generate quadrilateral elements directly include the paving method. The paving method was first proposed by Blacker *et al.* in 1990 [90] and was patented by Blacker in 1994 [91]. It is similar to the advancing front method but directly generates quadrilateral elements. For a detailed explanation of how the method works, the author suggests [61]. The paving method will not be discussed further here, as the patent would restrict its use within this work.

It has been suggested that several meshing techniques can be combined to create a hybrid mesh. These hybrid meshes may be applied independently to different surfaces within a single model [92], or on a single surface to accelerate mesh generation or to improve the mesh quality [83, 93]. These hybrid meshes may combine both structured and unstructured techniques. This is attractive for BE analysis since it may be more efficient if different shaped faces on the model are meshed in different ways. This becomes even more powerful if it is realised that for BE analysis the nodes around the edges of these faces do not necessarily need to coincide as the BEM will work equally well with discontinuous elements.

Lo and Lee [93] use a mixed meshing technique, which they call coring, to improve the speed of the mesh generator. A regular grid of square elements is generated over the interior of a model, covering as much area as possible without intersecting the boundary. The space around this core is then filled by means of a Delaunay triangulation. El-Hamalawi [83] uses a combined advancing front-Delaunay mesh generator to improve the speed of the process by removing the hazard of creating overlapping elements during the final pass of the advancing front technique. Kallinderis and Shaw [92, 94] also use hybrid grids to increase the efficiency of the mesh.

3.5 Mesh grading and refinement

To generate an optimal mesh, a sophisticated grading system is required to ensure the mesh is refined appropriately around points of key interest. This requires a detailed mesh in areas of high stress gradient and a coarser mesh elsewhere to minimise computational effort. There are several approaches to mesh optimisation:

- A coarse mesh is generated over the entire model and then refined in regions of high stress gradient.

- A fine mesh is generated over the entire model and then coarsened in unimportant areas.
- A background mesh or set of equations is used to define the element size over the model and the mesh generated to satisfy these rules.

Error estimators are often used to inform refinement of a mesh. These require one, or multiple analysis runs to be carried out before they can be applied. There are two main families of error estimators: *A priori* methods [95] look at the convergence rate of the results as a model is successively analysed whilst increasing the mesh density. These method provide a measure of the efficiency of a given method but cannot be used to inform mesh refinement. *A posteriori* techniques, which are extensively reviewed by Grätsch and Bathe [96], give a measure of the accuracy of analysis results and are therefore commonly used to inform mesh refinement. These adaptive refinement techniques [97] require that an initial analysis is run on a coarse mesh. The mesh is then refined in areas that feature a large stress gradient suggested by an error indicator. The most commonly applied error indicators are *superconvergent patch recovery* after Zienkiewicz and Zhu [98, 99] in the FEM and the *energy norm* [100] in the BEM. If the model is successively analysed, the errors assessed and the mesh refined accordingly, the mesh will converge towards an optimal form. For speed, it should be possible to generate a suitable mesh based on the initial model geometry or loading conditions without the need for further refinement.

Shepherd *et al.* [101] coarsen a quadrilateral mesh by collapsing and removing surplus elements. The nodes around the elements to be removed are moved towards each other along the element boundary until they meet and are combined, thus collapsing the element. Topping *et al.* [61] suggest generating a crude background mesh over the model. The background mesh is created by the analyst and is used to define the density of the mesh over the entire model. A variable is defined by the background mesh that specifies the distance between the nodes defining an element generated at that point. This variable can be constant within the defined region or governed by an equation. For example, the variable may be altered depending on the distance between a given element and a point in the background mesh.

Xu *et al.* [84] combine all three approaches to grade their mesh. An initial mesh is generated and a background mesh is applied to inform the mesh generation algorithm where the mesh can be refined or coarsened.

Lo [79] proposes a scheme which uses the segments initially generated around the model boundary to define the element size across the interior. A simple algorithm is used to define the internal node spacing based on the distance of each node from the boundary and the size of the segments on the boundary. The elements around the boundary must be suitably refined first but this can be done simply through geometric recognition and basic interpolation.

3.6 Storage schemes and data structures

Data defining the elements in the model can be stored in different ways. Element based and node based data structures store and access data through a traversal of the elements or nodes.

Wang *et al.* [102] use a node based storage scheme for generation of quadrilateral meshes. This scheme defines the model purely through the interconnectivity of nodes. If an element is required it is constructed through a traversal of the nodes at its corners, starting with the root node, which lies at the same corner of all the elements in the model. Each node stores pointers to its neighbours. This results in a reduced memory requirement over the element based data structure.

Topping *et al.* [61] use an element based data structure for their Delaunay triangulation. Their *triangle matrix*, which defines the elements and interconnectivity, incorporates the node numbers that form the corners of the element, the numbers of the adjacent elements, which element shares each side and a series of indicator flags. These parameters can be used to find the interconnectivity of the entire

mesh with little time consuming searching of matrices.

A mesh can be fully defined using only the nodal coordinates and an array defining the nodal connectivity. However, if memory is available, extra details may be stored allowing increased access speed to all of the data. Node based data structures typically use less memory but require more central processing unit (CPU) time for the additional traversals required.

When using the BEM it is important to define clearly where the object material is located. This definition will make the difference between modelling a solid object, such as a cube, or modelling a cube shaped hole in an infinite solid. When a surface is viewed from outside, the external boundaries must be orientated in an anticlockwise manner so that the interior of the surface is always to the left of the boundary. This means that any internal boundaries must be clockwise orientated. The cross product of the surface outward normal and any vector pointing along the boundary always points towards the interior of the surface. The same rule is used to define the orientation of the elements.

3.7 Meshing for re-analysis

A model may be re-analysed because the original results are not accurate, generally due to a insufficiently refined mesh, or because the model geometry has changed.

Chellamuthu and Ida [76] describe a mesh refinement strategy based on an analysis of the initial mesh. Areas that need to be refined, located by calculating errors in the result, are iteratively marked, re-meshed and re-analysed until a predefined tolerance is satisfied. This is a commonly used procedure for adaptive mesh refinement. Re-meshing conventionally refers to regeneration of the mesh over the entire domain as part of an adaptive scheme to reduce errors in the analysis. This is a different procedure to that which will be utilised in this thesis to update the mesh to account for changes in the model geometry.

For real-time updating of the model due to a change in geometry, it is possible to refresh parts of the mesh only in the areas where alterations to the mesh are required. Previously this has primarily been applied to flow problems and usually requires the entire mesh be checked systematically for changes. Yiu *et al.* [15] demonstrate the adaptivity of the quadtree method when applied to moving fluid flow. Areas of the mesh can easily be regenerated based on the surface motion of a wave or similar. This surface is discretised to generate additional seeding points, used when adapting the mesh.

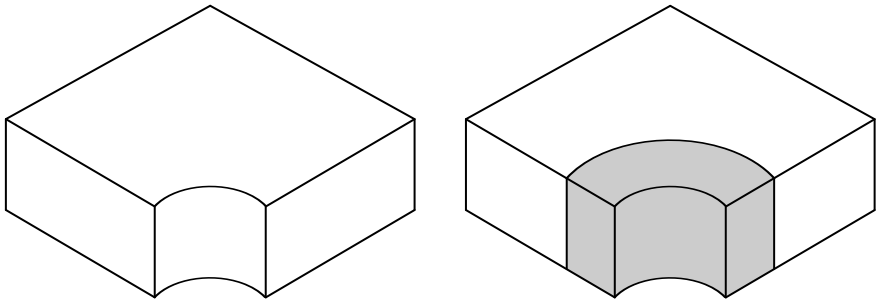
Shimada and Gossard [88] recommend the bubble packing method as a powerful system when continuous local re-meshing is required due to changing surface geometry. The initial configuration may be used as a starting point and only a relatively small number of iterations are required to redefine the mesh. Only bubbles within the area of updated geometry need to be reprocessed leaving the majority of the mesh intact. This also reduces the time required to update the system matrix, as most of it remains unchanged.

Aubry *et al.* [103] use an advancing front scheme, coupled to a background grid, to refine an existing surface mesh to reduce errors in analysis or for graphical displays.

Zoning is a common technique in the BEM which is used to separate models into several regions or zones as shown in Figure 3.12. This is required when parts of the model are made of different materials. It can also be used to break models into appropriate regions where the global geometry is unsuited to the BEM. Breaking the model into zones also enables them to be independently re-meshed. This removes the need to update the entire mesh each time a change in model geometry affects just one region. For example, in Figure 3.12(b), if the radius of the circular hole were to be changed, only the mesh within the grey zone would need to be regenerated. The mesh over the white zone (and hence the integrals associated with this region) would remain unchanged. However, a compromise would have to be reached between the size of the zone and the size of perturbation that can be applied to the updated geometry as

a large update could affect the geometry outside the zone. Geometric recognition algorithms would be required to decide which areas of geometry are most likely to be updated and hence where to construct separate zones.

Zoning can also break up the system matrix by introducing coarse banding. This saves disk space and simplifies the process of updating this matrix, increasing the speed of the process. Trevelyan [30] discusses zoning for efficiency where it is used to reduce the amount of processing time required to build the system matrix. He also demonstrates how zoning should be used to produce a more accurate analysis.



(a) Single zone. (b) Two zones.

Figure 3.12: Zoning around features.

All the techniques so far discussed in this section are aimed primarily at improving the accuracy of an analysis. In this thesis, re-meshing will be used to accelerate the re-analysis of the model whilst maintaining an acceptable level of accuracy. Whilst Trevelyan [16] has extensively researched this approach in two dimensions, very few authors have previously addressed three-dimensional problems. Rendall and Allen [104] and Michler [18] use mesh deformation techniques to adapt an existing mesh around aircraft control surfaces after they have been moved to carry out a manoeuvre. This is done to preserve mesh connectivity and to reduce re-analysis times. Both authors use radial basis functions to accurately interpolate the new nodal positions from the surrounding nodes over complex geometries.

Surgical simulation uses deformable models to visually simulate, or to supply haptic feedback for, the manipulation and cutting of organs and tissue to aid in the training of surgeons. Wang *et al.* [9] use constant triangular elements with a single node at the centroid to simulate tissues. This requires a fine mesh for accurate analysis, however the number of degrees of freedom (DOF) are reduced by grading out to a coarse mesh in areas that are not being operated on. During simulation fewer than 10 elements are typically modified so that the majority of the mesh remains unchanged and feedback can be supplied in real-time. Other techniques often require a very crude initial mesh to facilitate speed.

3.8 Three-dimensional surface modelling

The majority of the articles discussed so far demonstrate meshing algorithms over purely planar surfaces. In reality surfaces are often curved. This requires that elements be generated over a more complex domain. The most common technique to overcome this problem is to map a plane mesh onto the three-dimensional surface [80, 105].

Lo [105] demonstrates a series of basic transformations which are used to triangulate meshes over simple and multiply-connected three-dimensional surfaces, such as parts of a cylinder or sphere. He proposes splitting the model so that each simple region can be meshed separately before being combined into a global mesh. Lo later extends this method to quadrilateral elements [106]. Peiró [107] uses a set of transformations to map a valid triangulation generated on a plane onto a surface in such a way that the triangulation remains valid.

Plane to surface mapping can lead to problems such as severe distortion of the elements. Hinton *et al.* [80] overcome this problem by splitting the surface into *Coons patches* and generating stretching factors to distort the two-dimensional mapping of these patches in such a way that the effects of distortion, when mapping back into three dimensions, are minimised.

Lau and Lo [66] go one-step further and generate elements directly over the model surface using an advancing front technique. The precise location of any analytical surface is dictated by a formula. Lau and Lo use this knowledge to develop a scheme for applying the advancing front technique directly to a surface. This has the benefit of being able to refine elements around the surface whilst maintaining their quality, rather than relying on a mapping that may result in distorted elements. The only drawback of the method is that the surface must be analytical. Lau *et al.* go on to extend this scheme to generate quadrilateral meshes by merging the triangular mesh [108].

In cases of extreme curvature, a mapped mesh may become so distorted that elements overlap. This causes serious problems with analysis. Wang and Tang [109] deal with this issue by adaptively defining a system of control vectors associated with the boundary of the region over which the mapping is to be carried out. This technique works for many surface types although it is computationally expensive and not infallible; the authors themselves identify cases where the mesh remains self-overlapping regardless of the optimisation carried out.

Three-dimensional surface representation can introduce inaccuracies over the entire surface in a similar manner to around two-dimensional curves, as discussed in Section 3.3. Elements need to be very small and of high order to model curved surfaces precisely, otherwise a potentially unacceptably large discretisation error may be introduced. Lau and Lo [66] introduce a scheme to minimise this error through careful control of the maximum angle between elements. Gelas *et al.* [110] compare several different surface meshing algorithms with a view to finding the method with the smallest discretisation error.

Many surface meshing algorithms have been developed purely to represent the surface graphically and often contain too much detail for efficient numerical analysis. Two of the primary applications for these algorithms are in geographical mapping software, where a highly accurate surface representation is required, and in computer games, where it does not matter if the object is accurately meshed so long as it appears correct. These schemes would be impractical and inaccurate for BE analysis and will not be discussed further here. If more information is required, the author recommends [110–112].

Re-meshing of three-dimensional surfaces is often used to improve the properties of the mesh and hence the quality and level of detail in the graphical display of three-dimensional objects [113, 114]. A concise review and classification of re-meshing techniques for graphical applications can be found in [115]. Alexa [116] presents a review of mesh morphing techniques to create a smooth transition between different three-dimensional meshes in computer graphics.

3.9 Concluding comments

Thompson *et al.* [117] summarise the process of generating a mesh when they state that: 'Grid generation is still something of an art, as well as a science'. Many of the algorithms formulated for mesh generation rely on the author's opinion of what makes a good mesh. This is always based on previous research that has explored many different systems and in most cases derived a satisfactory solution. Opinion is also divided on how to measure the quality of a mesh. Different systems of error analysis promote different meshing strategies and some mesh types are preferential to others when analysing particular problems. When generating a mesh there is still a large element of personal opinion on which is the best method to use. No scheme is perfect; however, the major mesh generation strategies are robust and reliable enough to provide a practical solution to most problems.

Chapter 4

Numerical integration

"Human beings, who are almost unique in having the ability to learn from the experience of others, are also remarkable for their apparent disinclination to do so."

Douglas Adams

4.1 Introduction

In this chapter the various Gaussian integration schemes available to integrate over a boundary element mesh formed from triangular and quadrilateral continuous, quadratic serendipity elements are discussed. These include both standard integration schemes and techniques for near-singular integrals.

The adaptive cross approximation (ACA) technique is a method for compressing the boundary element (BE) system matrix $[A]$ and is commonly used to accelerate the solution of a linear system of equations. However, if a partial ACA scheme is employed it may be possible to also accelerate the integration and re-integration phases. Section 4.3 discusses the benefits and drawbacks of the ACA scheme when applied to the integration.

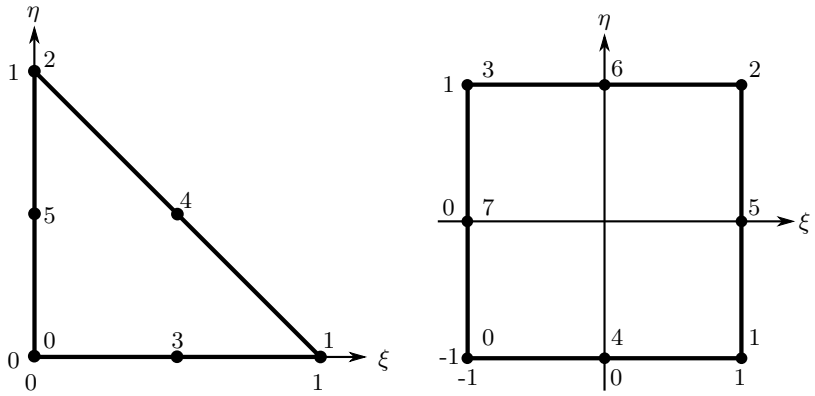
4.2 Integration

Numerical integration techniques are used to integrate the traction and displacement kernels over each element that makes up a mesh. Several different techniques must be applied to enable the integration to be computed. These are summarised in this section.

4.2.1 Interpolation

For isoparametric elements, a set of functions, N_k where k is the node number, are used to interpolate both the geometry and the solution variables (displacement and traction) over the element. These are known as interpolation or shape functions and are defined on the parametric form of each element shown in Figure 4.1 which use the standard node numbering conventions. The local coordinate system, (ξ, η), is used to construct the shape functions. For three-dimensional triangular BEs, $0 \leq \xi \leq 1$ and $0 \leq \eta \leq 1$. For three-dimensional quadrilateral BEs, $-1 \leq \xi \leq 1$ and $-1 \leq \eta \leq 1$.

The isoparametric shape functions, first proposed by Irons and Zienkiewicz [118], described in this section are the standard quadratic shape functions for triangular and quadratic elements. As these functions are used to interpolate displacements, the shape functions also become the set of basis functions in which the solution is sought.



(a) Triangular element. (b) Quadrilateral element.

Figure 4.1: Parametric elements.

For the isoparametric quadratic triangular element given in Figure 4.1(a) the shape functions used in this work are:

$$N_0 = (1 - \xi - \eta)(1 - 2\xi - 2\eta) \tag{4.1}$$

$$N_1 = \xi(2\xi - 1) \tag{4.2}$$

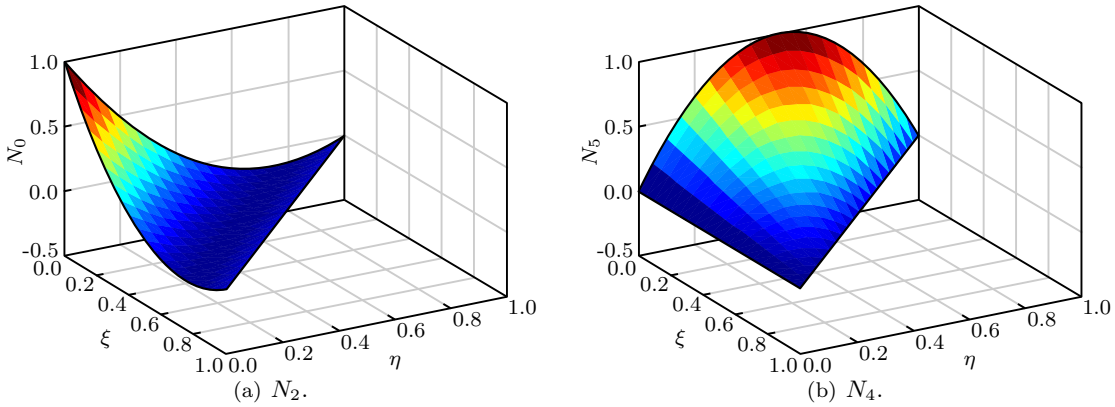
$$N_2 = \eta(2\eta - 1) \tag{4.3}$$

$$N_3 = 4\xi(1 - \xi - \eta) \tag{4.4}$$

$$N_4 = 4\xi\eta \tag{4.5}$$

$$N_5 = 4\eta(1 - \xi - \eta) \tag{4.6}$$

The surface created by N_2 is shown in Figure 4.2(a) and the surface created by N_4 in Figure 4.2(b). The surfaces created by the shape functions for the other corner nodes, N_0 and N_1, and mid-side nodes, N_3 and N_5, are similar to Figures 4.2(a) and 4.2(b) respectively but rotated so the peak value is at the subscript node.



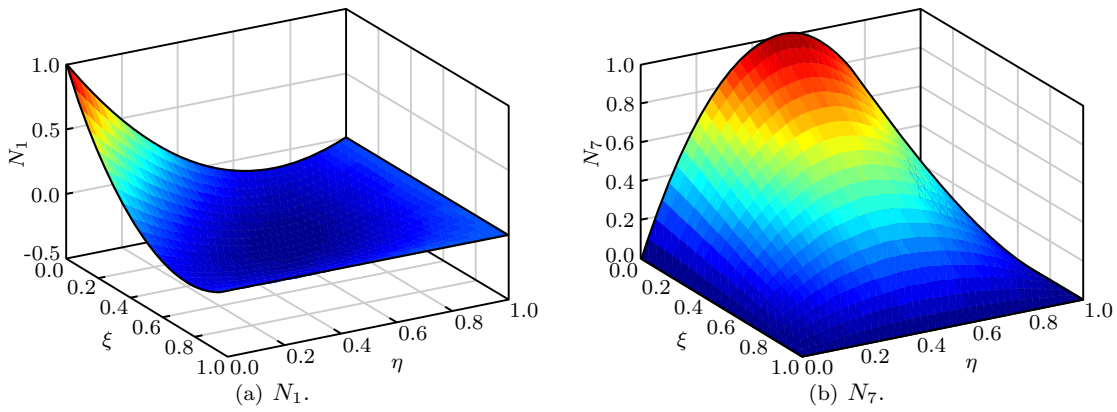
(a) N_2. (b) N_4.

Figure 4.2: Triangular quadratic shape functions.

For the isoparametric quadratic quadrilateral element given in Figure 4.1(b) the shape functions used in this work are:

$$N_0 = -0.25(1-\xi)(1-\eta)(1+\xi+\eta) \tag{4.7}$$
$$N_1 = -0.25(1+\xi)(1-\eta)(1-\xi+\eta) \tag{4.8}$$
$$N_2 = -0.25(1+\xi)(1+\eta)(1-\xi-\eta) \tag{4.9}$$
$$N_3 = -0.25(1-\xi)(1+\eta)(1+\xi-\eta) \tag{4.10}$$
$$N_4 = 0.5(1-\xi^2)(1-\eta) \tag{4.11}$$
$$N_5 = 0.5(1-\eta^2)(1+\xi) \tag{4.12}$$
$$N_6 = 0.5(1-\xi^2)(1+\eta) \tag{4.13}$$
$$N_7 = 0.5(1-\eta^2)(1-\xi) \tag{4.14}$$

The surface created by N_1 is shown in Figure 4.3(a) and the surface created by N_7 in Figure 4.3(b). The shape functions for the other corner nodes, N_0, N_2 and N_3, and mid-side nodes, N_4, N_5 and N_7, produce identical surfaces to Figures 4.3(a) and 4.3(b) respectively but rotated so that the peak value is at the subscript node.



(a) N_1. (b) N_7.

Figure 4.3: Quadrilateral quadratic shape functions.

4.2.2 Gauss-Legendre quadrature

The Gauss-Legendre quadrature method of integration uses a set of non-uniform integration, or Gauss, points spread across each element. The location of these points is designed for integration of polynomial functions, though they can be used for integration of any smooth, bounded function. Gauss-Legendre integration will produce an exact result for polynomials of order $2m-1$ where m is the number of Gauss points along each axis of the parametric element.

A weight must also be applied at each point. In two dimensions the method can be summarised as:

$$\int_{-1}^{1}\int_{-1}^{1} f(x,y)dxdy \approx \sum_{i=1}^{m_1}\sum_{j=1}^{m_2} f(x_i,y_j)w_{ij} \tag{4.15}$$

where w_{ij} is the weight associated with Gauss point ij. Suggested locations and weights for the Gauss points across each element are given in Sections 4.2.4 and 4.2.5.

4.2.3 The Jacobian

A transformation is needed to convert between the parametric elements and the Cartesian elements that make up the model. In two dimensions, this is given as:

$$J(\xi,\eta) = \frac{\partial(x,y)}{\partial(\xi,\eta)} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix} \tag{4.16}$$

Using J to transform into the parametric domain the following relationship can be derived:

$$\int_{-1}^{1}\int_{-1}^{1} f(x,y)dxdy = \int_{-1}^{1}\int_{-1}^{1} f(\xi,\eta)J(\xi,\eta)d\xi d\eta \approx \sum_{i=1}^{m} f(\xi_i,\eta_i)|J|w_i \tag{4.17}$$

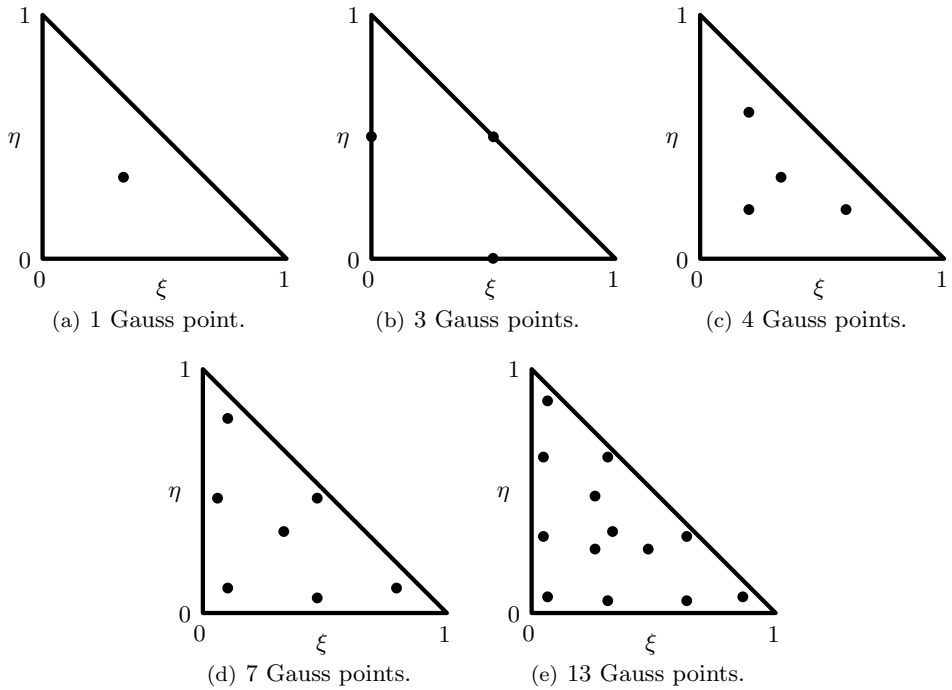
where $|J|$ is the determinant of the Jacobian:

$$|J| = \frac{\partial x}{\partial \xi}\frac{\partial y}{\partial \eta} - \frac{\partial x}{\partial \eta}\frac{\partial y}{\partial \xi} \tag{4.18}$$

4.2.4 Standard integrals

Triangular elements

Five different integration schemes for triangular elements are presented here. These are summarised in Figure 4.4 and Table 4.1. It should be noted that the weights given in Table 4.1 are half those quoted in most literature; this is due to the fact that the sum of the weights should equal the area of the parametric element. Normally they would be multiplied by a factor of 0.5 during computation of the integrals.



(a) 1 Gauss point. (b) 3 Gauss points. (c) 4 Gauss points.

(d) 7 Gauss points. (e) 13 Gauss points.

Figure 4.4: Triangular element integration schemes.

More detailed integration schemes may be constructed by splitting the triangle into four equal sub-elements as shown in Figure 4.5 and applying one of the schemes given in Figure 4.4 to each sub-element.

Table 4.1: Triangular element integration schemes.

$\hat{m}$	ξ	η	w
1	0.3333333333	0.3333333333	0.5000000000
3	0.5000000000	0.5000000000	0.1666666667
	0.0000000000	0.5000000000	0.1666666667
	0.5000000000	0.0000000000	0.1666666667
4	0.3333333333	0.3333333333	-0.2812500000
	0.6000000000	0.2000000000	0.2604166667
	0.2000000000	0.6000000000	0.2604166667
	0.2000000000	0.2000000000	0.2604166667
7	0.1012865073	0.1012865073	0.0629695903
	0.7974269854	0.1012865073	0.0629695903
	0.1012865073	0.7974269854	0.0629695903
	0.4701420641	0.0597158718	0.0661970764
	0.4701420641	0.4701420641	0.0661970764
	0.0597158718	0.4701420641	0.0661970764
	0.3333333333	0.3333333333	0.1125000000
13	0.0651301029	0.0651301029	0.0266736178
	0.8697397942	0.0651301029	0.0266736178
	0.0651301029	0.8697397942	0.0266736178
	0.3128654960	0.0486903154	0.0385568805
	0.6384441886	0.3128654960	0.0385568805
	0.0486903154	0.6384441886	0.0385568805
	0.6384441886	0.0486903154	0.0385568805
	0.3128654960	0.6384441886	0.0385568805
	0.0486903154	0.3128654960	0.0385568805
	0.2603459661	0.2603459661	0.0878076287
	0.4793080678	0.2603459661	0.0878076287
	0.2603459661	0.4793080678	0.0878076287
	0.3333333333	0.3333333333	-0.0747850223

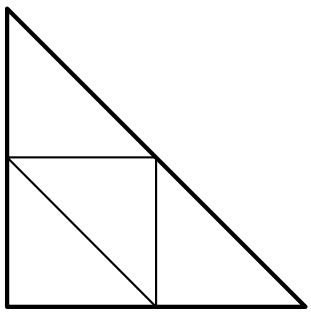
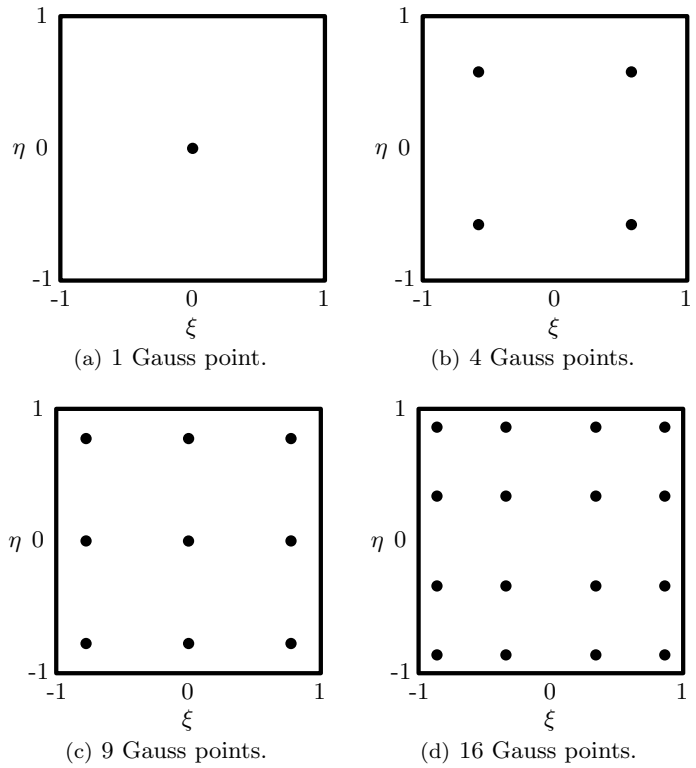


Figure 4.5: Triangular sub-elements.

Quadrilateral elements

Four different integration schemes for quadrilateral elements are presented here. These are summarised in Figure 4.6 and Table 4.2. More detailed integration schemes may be constructed by dividing the element into four equal sub-elements and applying one of the schemes given in Figure 4.6 to each sub-element.



(a) 1 Gauss point. (b) 4 Gauss points.

(c) 9 Gauss points. (d) 16 Gauss points.

Figure 4.6: Quadrilateral element integration schemes.

4.2.5 Singular integrals

Singular integrals occur in the boundary element method (BEM) when a field element is integrated relative to a source node that lies on the same element. The displacement kernel, U_{ij}, given in equation (2.18), contains the factor $1/r$, where r is the distance between the source and field nodes, and is therefore weakly singular as $r \to 0$. The traction kernel, T_{ij}, given in equation (2.21), contains the factor $1/r^2$ and is therefore strongly singular. When $r = 0$ these singular integrals cannot be computed directly. However, the near-singular integrals, which occur when the source and field points lie on the same element but are not the same point, can be calculated using a more detailed and appropriate integration scheme. The singular terms are found later through rigid body motion (see Section 4.2.6).

The weights and Gauss point locations associated with the near-singular integration schemes described in this section are constructed over the entire element rather than using sub-elements. Elements containing a singularity may therefore be integrated in exactly the same way as the non-singular elements. All integrals are therefore compatible with a reusable intrinsic sample point (RISP) integration scheme [29].

Triangular elements

The near-singular integration schemes for triangular elements are constructed by splitting the element into sub-elements about the singular node. Each sub-element is treated as a quadrilateral element with two nodes lying at the singularity. This mapping removes the singularity. Figure 4.7 gives example

Table 4.2: Quadrilateral element integration schemes.

$\hat{m}$	ξ	η	w
1	0.0000000000	0.0000000000	4.0000000000
4	-0.5773502692	0.5773502692	1.0000000000
	-0.5773502692	-0.5773502692	1.0000000000
	0.5773502692	-0.5773502692	1.0000000000
	0.5773502692	0.5773502692	1.0000000000
9	-0.7745966692	-0.7745966692	0.3086419754
	-0.7745966692	0.0000000000	0.4938271605
	-0.7745966692	0.7745966692	0.3086419754
	0.0000000000	-0.7745966692	0.4938271605
	0.0000000000	0.0000000000	0.7901234568
	0.0000000000	0.7745966692	0.4938271605
	0.7745966692	-0.7745966692	0.3086419754
	0.7745966692	0.0000000000	0.4938271605
	0.7745966692	0.7745966692	0.3086419754
16	-0.8611363116	-0.8611363116	0.1210029933
	-0.8611363116	-0.3399810436	0.2268518518
	-0.8611363116	0.3399810436	0.2268518518
	-0.8611363116	0.8611363116	0.1210029933
	-0.3399810436	-0.8611363116	0.2268518518
	-0.3399810436	-0.3399810436	0.4252933031
	-0.3399810436	0.3399810436	0.4252933031
	-0.3399810436	0.8611363116	0.2268518518
	0.3399810436	-0.8611363116	0.2268518518
	0.3399810436	-0.3399810436	0.4252933031
	0.3399810436	0.3399810436	0.4252933031
	0.3399810436	0.8611363116	0.2268518518
	0.8611363116	-0.8611363116	0.1210029933
	0.8611363116	-0.3399810436	0.2268518518
	0.8611363116	0.3399810436	0.2268518518
	0.8611363116	0.8611363116	0.1210029933

sub-element and Gauss point locations. These use two sub-elements with 16 Gauss points each if the singularity lies on a corner of the element and four sub-elements are used with 9 Gauss points each if the singularity is located at a mid-side node.

Quadrilateral elements

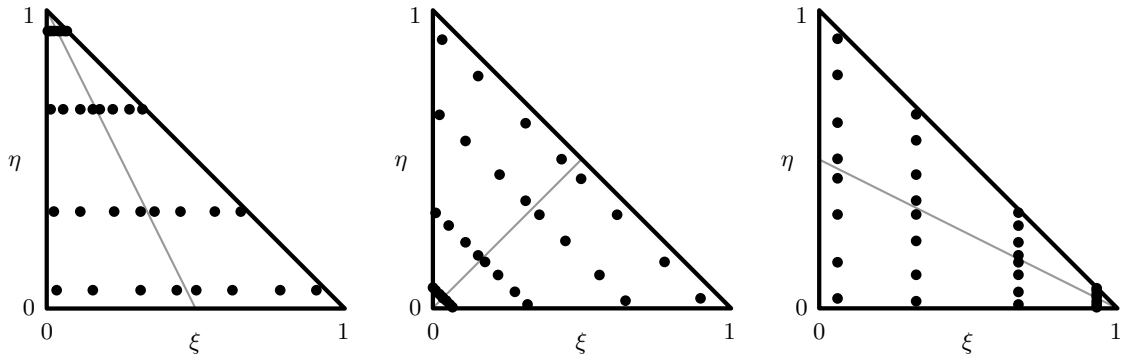
The near-singular integration schemes for quadrilateral elements are constructed in the same way as those for triangular elements, with four triangular sub-elements if the singularity is located at a corner node and six sub-elements if the singularity is located at a mid-side node. Example sub-element and Gauss point locations are shown in Figure 4.7. In this example, all the sub-elements contain 16 Gauss points.

4.2.6 Rigid body motion

All the non-singular and near-singular integrals are combined to form the linear system of equations:

$$[A]\{x\} = \{b\} \tag{4.19}$$

where $\{x\}$ is unknown and $[A]$ is fully-populated apart from the singular terms that lie in 3×3 blocks along the diagonal. If the model were to be moved in the x direction through a unit distance then $\{x\}$



(a) 32 Gauss points, singularity at node 0.

(b) 32 Gauss points, singularity at node 1.

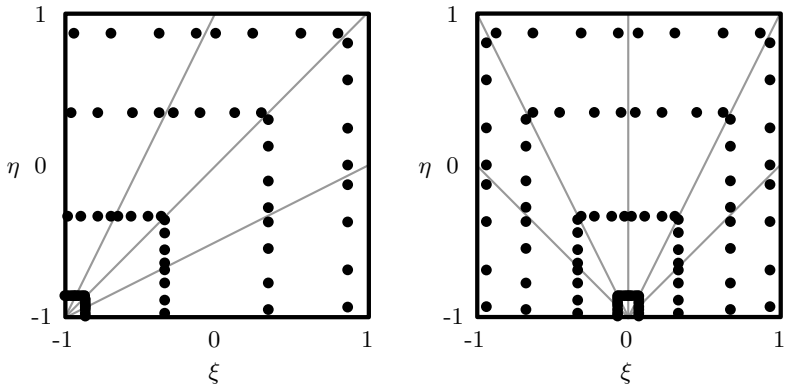
(c) 32 Gauss points, singularity at node 2.

(d) 36 Gauss points, singularity at node 3.

(e) 36 Gauss points, singularity at node 4.

(f) 36 Gauss points, singularity at node 5.

Figure 4.7: Singular triangular element integration schemes.



(a) 64 Gauss points, singularity at nodes 0, 1, 2 or 3.

(b) 96 Gauss points, singularity at nodes 4, 5, 6 or 7.

Figure 4.8: Singular quadrilateral element integration schemes.

would become $\{x_x\} = \{1\ 0\ 0\ 1\ 0\ 0\ \ldots\}^T$. Every third singular value in $[A]$ can now be found by solving:

$$\{A_{dx}\} = \{b\} - [A]\{x_x\} \tag{4.20}$$

If the model were translated in the y or z directions through the same unit distance then $\{x_y\} = \{0\ 1\ 0\ 0\ 1\ 0\ \ldots\}^T$ and $\{x_z\} = \{0\ 0\ 1\ 0\ 0\ 1\ \ldots\}^T$ can be used with the same method to compute $\{A_{dy}\}$ and $\{A_{dz}\}$. These terms can be used to fill in the rest of the terms in $[A]$.

4.2.7 Integration schemes

To improve the efficiency of boundary element integration, the literature suggests customising the integration scheme across each element depending on the distance, r, between the field element, e, and the current source node, p [119–121]. If r is large the variation in the stresses across e caused by a load at p will be small and hence a low order integration scheme can be applied. As r decreases, higher order schemes must be introduced, with specialist schemes used to deal with singular and near-singular integrals. As the scheme is specific to the current element-node pairing, the geometric data associated with the element - shape functions, Gauss point locations, weights and normals - are re-computed for every node in the model. Integration can then be carried out and the $[H]$ and $[G]$ sub-matrices corresponding to the current element-node pairing constructed. With the application of boundary conditions these are assembled to form the $[A]$ matrix and $\{b\}$ vector given in (2.34). The strongly singular terms on the diagonal of $[A]$ taken from $[H]$ are calculated by applying rigid body motion. The weakly singular terms from $[G]$ are already computed to sufficient accuracy using the near-singular integration scheme. The complete system is then passed to the linear solver.

The number of Gauss points, m, required to integrate over each dimension of an element is a function of the size, L, of the element and the minimum distance, R, between the element and the source node. In three dimensions, L is the length of the curve through the centre of the element in the integration direction as shown in Figure 4.9.

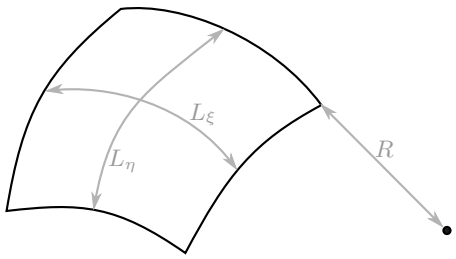


Figure 4.9: Measurements required to find m along both axes of a three-dimensional BE.

Lachat and Watson [119] developed an algorithm for finding m for quadrilateral elements. This was later generalised by Mustoe [120]. The original algorithm uses an iterative scheme to find the maximum ratio R/L; Gao and Davies [121] propose the following approximation to improve efficiency:

$$m = \frac{\lambda' \ln\left(\frac{e}{2}\right)}{2\ln\left(\frac{L}{4R}\right)} \tag{4.21}$$

where e is the prescribed tolerance of the relative integration error and

$$\lambda' = \sqrt{\frac{2}{3}\lambda + \frac{2}{5}} \tag{4.22}$$

where λ is the order of the singularity. It should be noted that this scheme assumes the integrand that

causes the singularity to be polynomial. A different formulation would be needed for log singularities, for example. The number of Gauss points suggested by the scheme is plotted in Figure 4.10. It should be noted that if $R/L < 0.25$ the integration order grows to infinity. If this is the case the element must be subdivided and the scheme applied to each sub-element.

An alternative scheme, based on numerical tests for surface integrals, is proposed by Bu and Davies [122]. This is generalised by Gao and Davies [121] and can be summarised as:

$$m = -0.1\lambda' \ln\left(\frac{e}{2}\right)\left(\left(\frac{8L}{3R}\right)^{\frac{3}{4}} + 1\right) \tag{4.23}$$

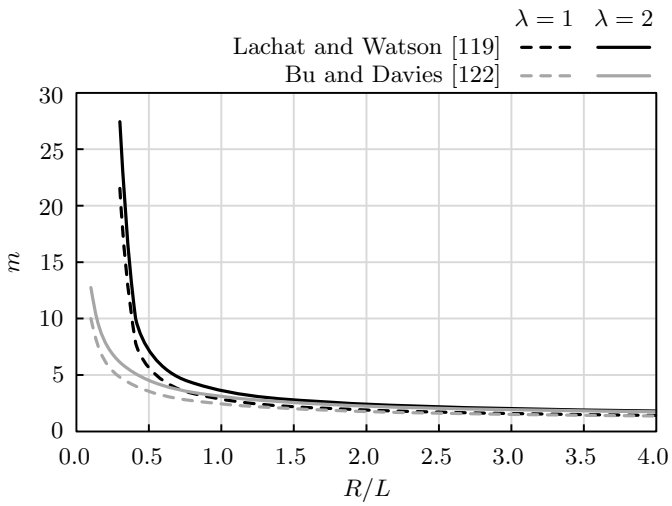


Figure 4.10: Integration order.

The standard BE integration scheme is shown in Figure 4.11.

4.2.8 RISP algorithm

The RISP integration scheme is illustrated in Figure 4.12. It uses a discrete number of integration schemes for each type of element. This means that the shape functions and derivatives for each integration scheme need only be calculated once for each element type.

Using a discrete number of integration schemes leads to a reduction in memory requirements and, as the models encountered in this work are assumed to contain relatively few elements, it is possible to store all the geometric data associated with the integration in memory. The Gauss point locations and associated normals and Jacobians for all integration schemes are therefore only computed once for each element in the model. This is then stored for use in future operations. The singular and near-singular integration schemes are formulated such that they can be applied using the same algorithm as the standard integrals. Aside from this, the rest of the integration and construction of the linear system is carried out in the same manner as the standard integration scheme.

4.2.9 Suppression of integrals

The majority of the traction or displacement boundary conditions applied to a model have zero magnitude. These will appear as zeros in the corresponding cells in the $\{t\}$ and $\{u\}$ vectors found in equation (2.32). During construction of equation (2.34), these zero values multiply some of the integral equations. It is therefore not necessary to compute these integrals, thereby reducing the total integration time.

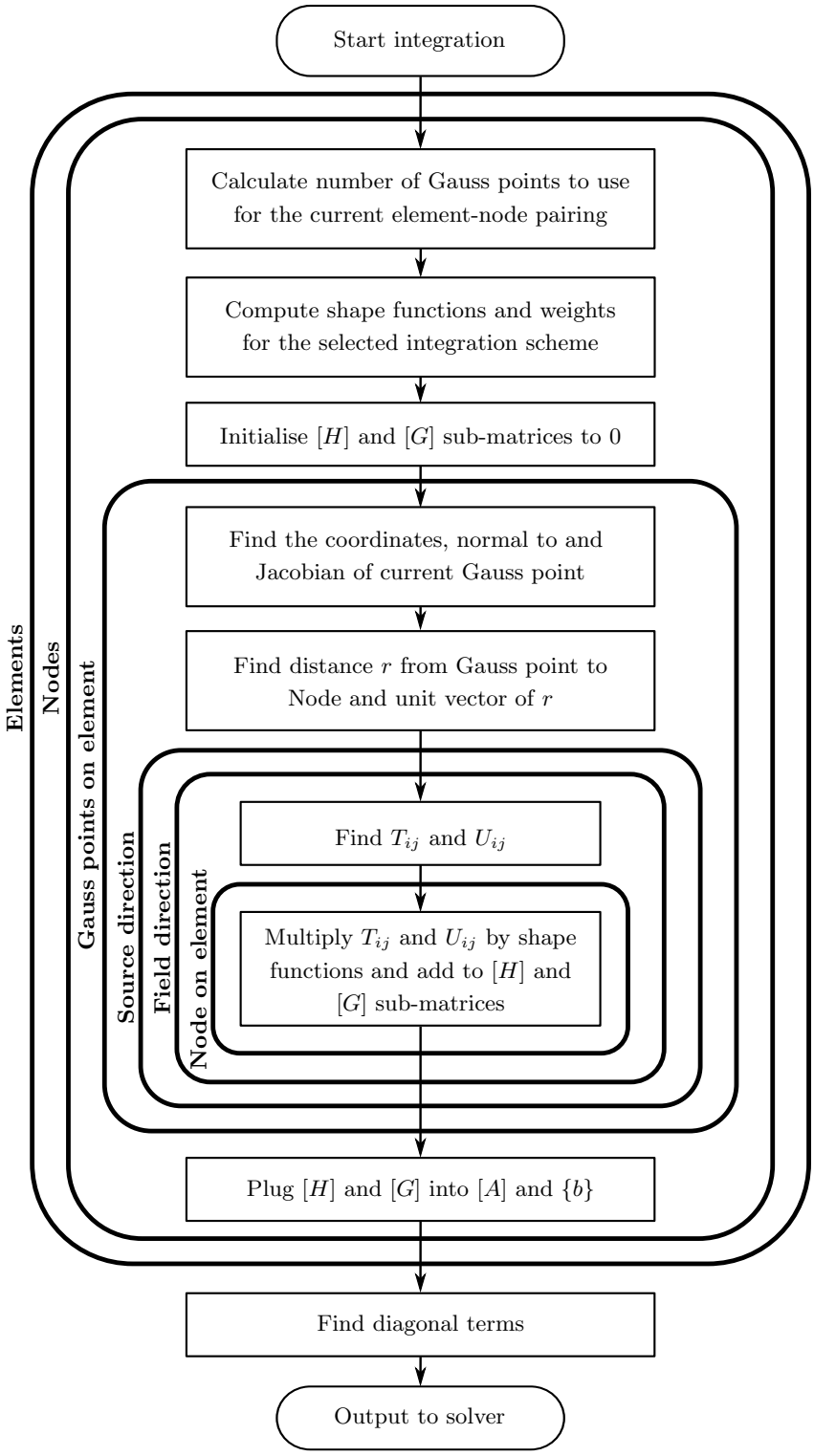


Figure 4.11: Standard integration flowchart.

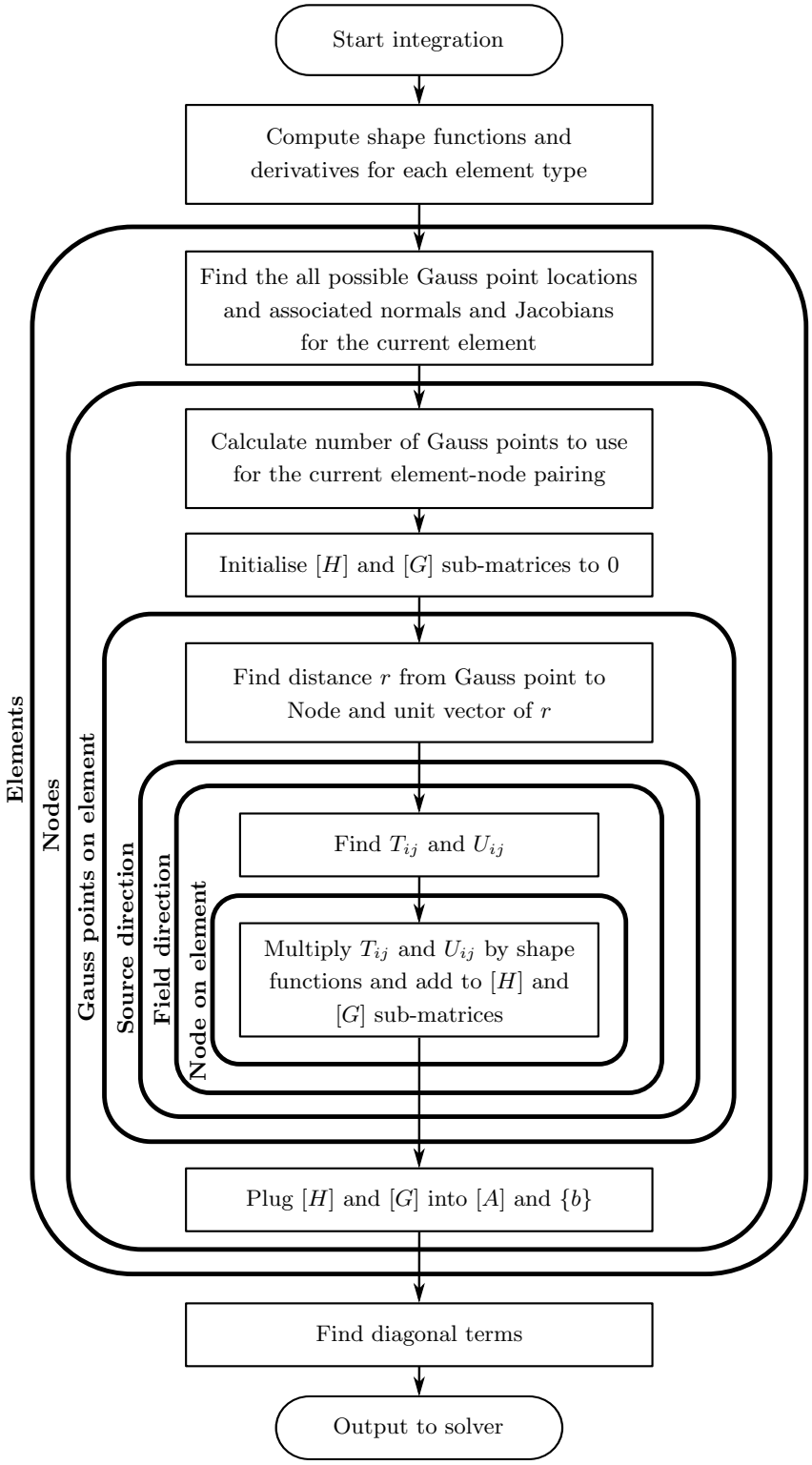


Figure 4.12: RISP integration flowchart.

4.2.10 Parallelisation

Due to the structure of the algorithm discussed in this work, as shown in 4.12, it can be easily parallelised by sharing the outer loop around multiple processors. It should be noted that the outer loops over elements and nodes are interchangeable.

4.3 Adaptive cross approximation

4.3.1 Introduction

In a system of linear equations:

$$[A]\{x\} = \{b\} \tag{4.24}$$

generated using the BEM, the matrix, $[A]$, is fully-populated and is therefore slow to assemble and requires a large amount of memory to store. To reduce these requirements it is desirable to approximate $[A]$ by compressing it into a reduced form. This will reduce the number of matrix terms that need to be computed and stored.

In the BEM, $[A]$ is filled with an array of data computed using kernel functions. Techniques such as the fast multipole method (FMM) [123], panel clustering [124] and $\mathcal{H}$-matrix methods [125], attempt to reduce the time to solve equation (4.24) by approximating the kernel functions. Although these methods have been shown to be effective, they still have some drawbacks, primarily the need to restructure completely the code used to construct equation (4.24).

Mosaic-skeleton methods generally use known values in $[A]$ to approximate the rest of the matrix. The algorithm can therefore be easily built on top of existing code. ACA [126] is one such approach. ACA assumes that areas of $[A]$ are analytically smooth, that is the change between consecutive values in $[A]$ is small. These areas are called admissible blocks. These can be represented as a series of vectors that can be cross multiplied to reconstruct an approximation to a block of $[A]$. The larger the number of vectors the more accurate the approximation.

An overview of the standard ACA techniques can be found in [127] along with algorithms for the implementation of block partitioning schemes and low rank matrix algebra.

ACA is typically used to compress large BEM system matrices, thereby reducing the memory requirements, and to reduce the run time of iterative solvers for a single analysis where the number of degrees of freedom (DOF), n, is large. This work is focussed on rapid re-analysis of small systems, $n < 5,000$, with continuously changing geometry. ACA has not yet been assessed for applicability to this kind of problem and could potentially be used to accelerate adaptive analysis, optimisation problems and updating schemes.

The fully pivoted ACA is commonly used to speed up analysis and reduce memory requirements. The partially pivoted ACA is rarely used although, if applied intelligently, it can be used to reduce the number of system matrix entries that need to be calculated. For small systems calculating these terms has been shown to be the major contributor to analysis and re-analysis time [128].

4.3.2 Literature review

ACA was initially proposed by Bebendorf [126], in the fully pivoted form (which requires calculation of every term in the system), as a low-rank method of approximating asymptotically smooth blocks taken from large dense unstructured matrices. In this paper, Bebendorf applies ACA to a range of large systems ($16,000 < n < 202,000$) and shows good compression. In [129] Bebendorf and Rjasanow build on this early work and introduce a new algorithm for partitioning the matrix into a larger number of blocks. A later paper [130] parallelises the method.

Kurz *et al.* [131] introduce the partially pivoted ACA, applying it to reduce the number of terms that need to be calculated in the BEM system of equations for electromechanical analysis. Kurz achieves additional speed-up in the use of the iterative generalised minimal residual (GMRES) solver as the vector structure of the ACA approximation can be used to accelerate the matrix vector multiplication which is a dominant part of the GMRES algorithm.

Weber *et al.* [132] apply ACA to crack propagation simulation. The model is divided into blocks using a process similar to octree grid generation, whereby the entire model is encased in a cuboid that is successively subdivided into eight until a desired admissibility criterion is reached. The fully pivoted ACA is primarily used to compress the resulting system. Some reduction is observed in the analysis time although this is effectively annulled due to the additional time required to construct the approximation.

Frederix and van Barel [133] apply ACA to two-dimensional wave scattering problems. A direct method is employed to solve the approximated system, making use of a QR factorisation. Numerical examples show good accuracy but no specific timings are included, although good efficiency is reported due to a reduction in the number of integrals that need to be calculated.

Maerten [134] uses a parallelised ACA approach to model faults and fractures in structural geomechanics and shows good speed-up. However, to achieve a more accurate approximation, a non-negligible increase in computational time is reported. Mallardo *et al.* [135] apply ACA with a GMRES solver to reduce greatly the analysis time for for simulation of noise control in an aircraft cabin but do not comment on the accuracy of their algorithm.

4.3.3 Theory

Definitions

For clarity the following definitions have been observed in this thesis:

- Partition: A group of associated nodes.
- Block: An area of the system matrix defined where two partitions containing source and field nodes overlap.

Partitioning the matrix

The majority of authors use the same partitioning scheme as is applied to construct $\mathcal{H}$-matrices [125] to create a blocked matrix structure. This structures the matrix by looking at the location of the nodes in the computer model. Given a group of field points, C_1, and source points, C_2, which form a symmetric pair of blocks within the system matrix, $[A]$, the admissibility of the blocks can be established. An admissible block is likely to be analytically smooth due to the smoothness of the fundamental solution and can therefore be approximated using ACA.

If the diameter of a set of points, C, d_C, is defined as the maximal distance between any two points, p and q, in C:

$$d_C = \max_{p,q \in C} \|p - q\| \tag{4.25}$$

and the distance between two sets of points, C_1 and C_2, $r_{C_1C_2}$, is defined as the distance between the nearest points in C_1 and C_2:

$$r_{C_1C_2} = \min_{p \in C_1, q \in C_2} \|p - q\| \tag{4.26}$$

then the admissible condition is defined as:

$$\kappa \min(d_{C_1}, d_{C_2}) \leq r_{C_1C_2} \tag{4.27}$$

where:

$$\|p - q\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2} \tag{4.28}$$

and κ is a user-defined constant which is often set to 1. If the condition is failed then both blocks must either be subdivided further or calculated in full. The constant, κ, may be increased to ensure greater spacing between groups of nodes and hence a greater anticipated smoothness. However, increasing κ will also result in a larger number of blocks. The basic partitioning process for a two-dimensional problem is illustrated in Figure 4.13. Each time an admissibility check is failed the block is subdivided by splitting the largest constituent partition into two parts along the centre of the largest (x, y) dimension. For a three-dimensional problem the partition is subdivided by drawing a plane through the largest (x, y, z) dimension. It should be noted that, whilst the example in Figure 4.13 is partitioned until there is only one node remaining in each non-admissible block, this is not common and a stopping criterion is usually set at a prescribed minimum partition size.

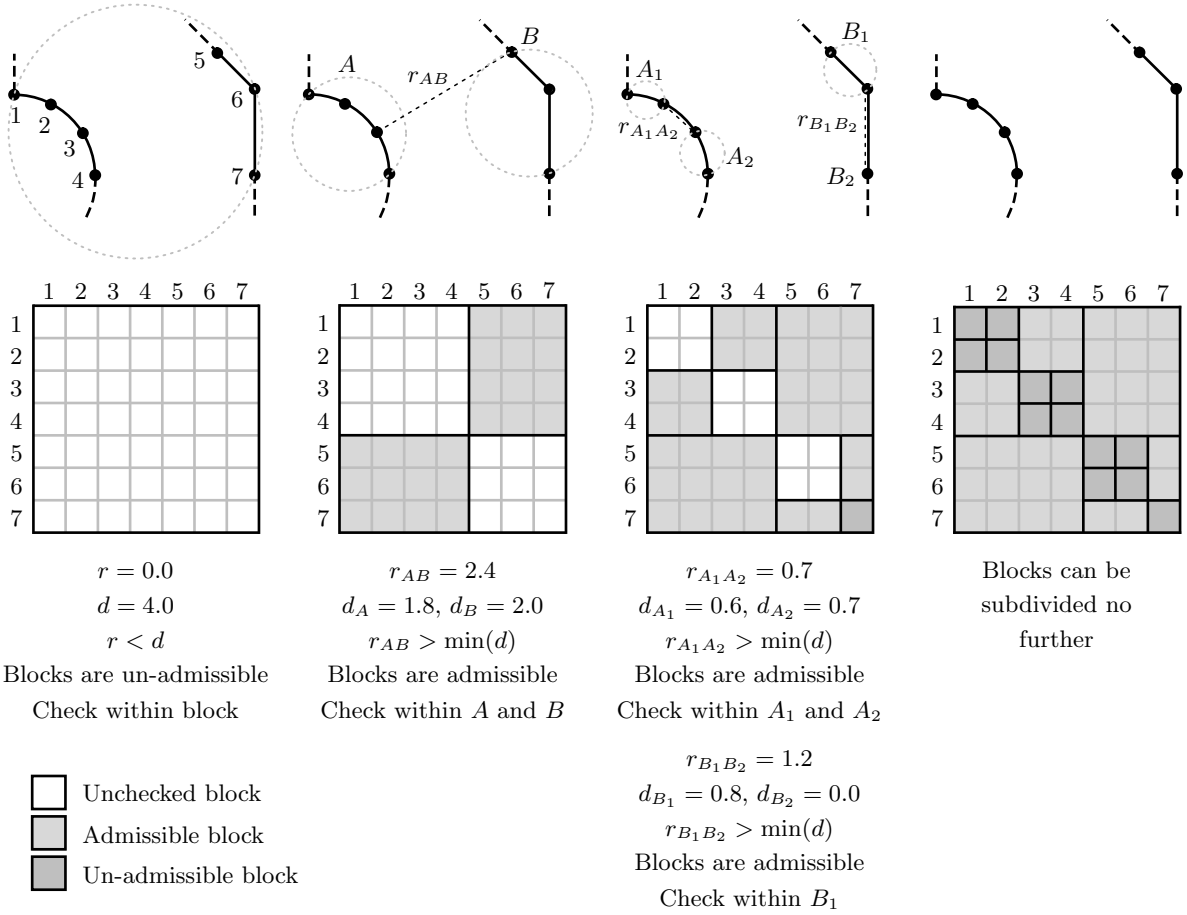
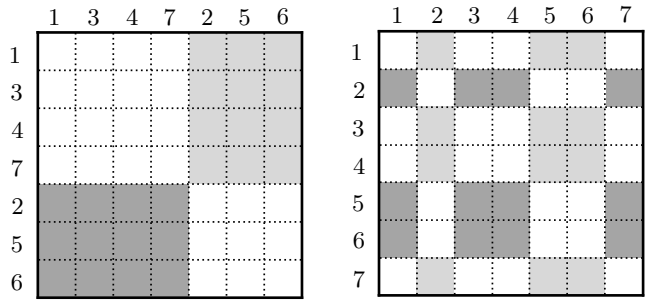


Figure 4.13: Partitioning the model, $\kappa = 1$.

Normally $\mathcal{H}$-matrix partitioning works by re-ordering the nodes in the model based on the location of the partitions. This means that block data are continuous, as shown in Figure 4.14(a). To avoid re-ordering $[A]$, a matrix of indices could also be constructed. This would enable each block to be split across several rows and columns of the matrix as shown in Figure 4.14(b).

Constructing the approximation

Two primary forms of the ACA algorithm exist. These are known as fully and partially pivoted ACA. The fully pivoted ACA is primarily a compression technique. It requires that each block of the full matrix



(a) Re-ordered matrix. (b) Indexed matrix.

Figure 4.14: Block structures.

be computed in full. The partially pivoted ACA uses knowledge of some of the terms in each block to approximate the rest of the block, thereby removing the need to compute every term.

Once the model has been partitioned, each admissible block, $[A']$, can be approximated using ACA. The remaining non-admissible blocks must be computed in full. In Algorithm 4.1, indices i and j respectively indicate rows and columns of $[A']$. When applying the fully pivoted algorithm, the values of i and j are determined by finding the term in the block of greatest magnitude. To construct a partially pivoted ACA, i is initially set to 1 and j is unknown. The index k counts the number of vectors, $\{u_k\}$ and $\{v_k\}$ $(k = 1, 2, ..., s)$, used to construct the approximation, $[S]$. The vector $\{e_i\}$ is a vector containing entirely zeros except for the ith entry, which contains 1. Arrays $\imath$ and $\jmath_k$ contain lists respectively of all the rows and columns of $[A']$ that have been used in construction of the approximation.

Algorithm 4.1 Partially pivoted ACA

$\imath_1 = i = k = 1$
$\{v_1\} = [A']^T\{e_1\}$
$\jmath_1 = j = \text{argmax}|\{v_1\}|$
$\{u_1\} = [A']\{e_j\}$
$\gamma = 1/\{v_1\}_j$
$\{v_1\} = \gamma\{v_1\}$
$[S_1] = \{u_1\}\{v_1\}^T$
while $\varepsilon_F > tol$ **do**
  $k = k + 1$
  $\imath_k = i = \text{argmax}_{i \notin \imath}|\{u_{k-1}\}|$
  $\{v_k\} = [A']^T\{e_i\} - \sum_{l=1}^{k-1}\{u_l\}_i\{v_l\}$
  $\jmath_k = j = \text{argmax}_{j \notin \jmath}|\{v_k\}|$
  $\{u_k\} = [A']^T\{e_j\} - \sum_{l=1}^{k-1}\{v_l\}_j\{u_l\}$
  $\gamma = 1/\{v_k\}_j$
  $\{v_k\} = \gamma\{v_k\}$
  $[S_k] = [S_k] + \{u_k\}\{v_k\}^T$
end while

In the fully pivoted ACA algorithm, convergence is checked by comparing the *Frobenius norm*, $\|\cdot\|_F$, of the current approximation, $[S_k]$, to that of $[A']$:

$$\varepsilon_F = \frac{\|[S_k] - [A']\|_F}{\|[A']\|_F} \tag{4.29}$$

However, when applying the partially pivoted ACA not all the terms in $[A']$ are known and therefore consecutive approximations are compared by replacing $[A']$ with $[S_{k-1}]$ in equation (4.29).

The Frobenius norm is given by:

$$\|S\|_F = \sqrt{\sum_{i=1}^{n'} \sum_{j=1}^{m'} S_{ij}^2} \tag{4.30}$$

where $[S]$ is a matrix block of size $m' \times n'$. Calculating ε_F using this method requires $2n'm'$ operations and is the most computationally expensive part of the entire algorithm.

4.4 Look up tables

Trevelyan and Scales [136] use look-up tables (LUTs), to store pre-computed boundary integrals for a number of two-dimensional orientations of source points and field elements. Interpolation is used to estimate values that appear do not appear in the LUTs. Least squares fits may be applied to construct a surface from the values given in each LUT. The formula for this surface may be used to calculate the value of the integral, thereby reducing the memory requirements by avoiding storing all the data associated the full LUTs. However, LUTs and surface fits would prove far more complex to apply to the three-dimensional BEM as the extra dimension would result in significantly increased memory requirements due to the additional possible combinations of element orientations and dimensions.

LUTs may also be used to store data on pre-solved models, such as those generated by Cotin *et al.* [137] to store model deformations and supply haptic feedback for surgical simulation. These LUTs are superimposed, via a modified superposition technique, to give a representation of the deformed geometry and thereby enable the integration and solution of the stiffness matrix to be bypassed. No stress feedback is computed and the goal of the research is not to compute an exact deformation, just to get the correct feel of prodding the tissue for training purposes. To enable feedback to be received in real-time, only elastic models are used, no re-meshing is carried out and deformations are considered to be small.

Wang *et al.* [10] maintain a constant number of elements as a model is deformed. This enables LUTs of pre-calculated solutions to be generated prior to simulation. A hypothetical unit displacement is applied, in turn, in the x, y and z directions at every node in the model and the resulting displacements and tractions are stored. An additional result is included for self weight. When the model is deformed, the required pre-calculated criteria are scaled accordingly and superimposed to find the actual solution. The set up time for this method is large but real-time feedback can be achieved during simulation. As the same model is often reused many times for different simulation routines, the LUTs only need to be calculated once and can then be stored for future use. Wang *et al.* only test small systems of around 500-800 DOF but infer from the small re-analysis time (less than 10^{-7} seconds) that much larger simulations should be possible.

Chapter 5

Linear equation solvers

"To know that we know what we know, and to know that we do not know what we do not know, that is true knowledge."

Confucius

5.1 Introduction

By re-writing equation (2.34) the system of n linear equations can be expressed as:

$$[A_i]\{x_i\} = \{b_i\} \tag{5.1}$$

where $i = 0, 1, 2, ...$ and refers to the number of times the system has been modified.

Every time the model is re-meshed the system is updated. As the majority of the changes are small the majority of matrix $[A_{i-1}]$ is preserved in $[A_i]$, where $[A_i] = [A_{i-1}] + [\Delta A]$. The matrix $[\Delta A]$ is sparse with only a few rows and columns containing non-zero values. The vector $\{b_i\}$ is entirely changed from $\{b_{i-1}\}$. The updated system must be re-solved to find the new tractions and displacements, $\{x_i\}$.

Kane *et al.* [3] develop two techniques for accelerating re-analysis of the boundary element method (BEM) which provide almost identical results to a complete re-analysis. The simple iteration re-analysis formulation will converge for small to moderate changes in the model and, if it converges, will be as accurate as a complete new analysis. The scaled, two-step iterative re-analysis formulation builds on the basic concept applied by Kirsch [138] to finite elements. The method features superior convergence characteristics when compared to the simple iteration technique due to application of constant scaling, whereby a scaling factor key to the convergence rate is calculated after the first iteration of the algorithm, or evolving scaling, whereby the scaling factor is recalculated during each iteration of the algorithm. This results in fewer iterations. However this does not necessarily make it faster as, due to the additional calculations, each iteration requires more central processing unit (CPU) time. Constant scaling may not converge for large changes and is generally slower than simple iteration techniques despite requiring fewer iterations. However, evolving scaling consistently outperforms the other methods. If multiple re-analyses are carried out, it is assumed that these are different perturbations of the initial model rather than an iterative update of the last model solved. This causes increasing numbers of iterations to be required for each subsequent re-analysis and ultimately to degradation of the results.

The simplest way to solve equation (5.1) is to employ a direct solver. In this work two direct solvers are introduced:

1. Gauss elimination
2. Lower-upper (LU) decomposition

Direct solvers are very robust but are computationally expensive. For rapid analysis, the model must be efficiently re-analysed as the geometry is updated. Using a direct solver would be too slow and alternative solvers must be considered. These techniques typically result in a reduction in the level of accuracy but, if the method is appropriate, this will be small.

Two different approaches have been considered for rapidly re-solving the system; iterative solvers and reduction techniques. Seven algorithms have been analytically compared for re-solving equation (5.1):

1. Generalised minimal residual (GMRES) [19]
2. Bi-conjugate gradient stabilised (BiCGSTAB) [139]
3. Transpose free quasi-minimal residual (TFQMR) [140]
4. Leu's reduction method [23]
5. Eigenvector based proper orthogonal decomposition (E-POD) [25]
6. Singular value decomposition based proper orthogonal decomposition (SVD-POD) [26]
7. Static condensation

Algorithms 1-3 are iterative solvers whilst 4-7 utilise reduction techniques.

5.2 Literature review

Several authors introduce solvers designed specifically for dense matrix systems such as those found in the BEM. Garcia *et al.* [141] introduce a new technique called matrix system reduction solution (MSRS). MSRS is designed to accelerate real-time solution of dense finite element (FE) systems for graphical display and as such is not concerned with the accuracy of the results so long as they provide a visually plausible result. Garcia *et al.* conclude that the solution is sufficiently accelerated but that the main constraint on the solution time is the time required to update the mesh. Kauers [142] introduces an existing, homomorphic images algorithm for fast solution of large, dense, non-symmetric systems with no special structure. They state that faster more recent specialist algorithms exist for structured matrices and that the one proposed is not the fastest for solving dense systems. These techniques will therefore not be considered further.

Barra *et al.* [143] present a study of the performance of the restarted GMRES algorithm when applied to BEM problems. The solver is compared with both Gauss elimination and a bi-conjugate gradient (BCG) solver, both with and without preconditioning. Barra *et al.* conclude that the preconditioned GMRES algorithm is up to 10 times faster than Gauss elimination and 2.5 times faster than the preconditioned BCG solver for small ($n < 1000$) two-dimensional boundary element (BE) problems. Tests on larger and three-dimensional problems are listed as future work. Kong *et al.* [144] also address the problem of finding a fast direct solver for the BEM. They break the system matrix into blocks and compute compressed factorisations of the inverse of the matrix, utilising column skeletons. The algorithm achieves good speed-up for models where $n > 1100$.

Trevelyan and Wang [145, 146] use an iterative GMRES solver based approach. The system matrix, $[A]$, is re-calculated with each re-analysis and a full matrix solution performed using the previous solution vector, $\{x_{i-1}\}$, as the starting point for the GMRES scheme. This requires similar numbers of iterations for each re-analysis. Trevelyan *et al.* [16] later go on to compare the different re-analysis approaches introduced by Kane [3] and Leu [23], Trevelyan and Wang [145, 146] and others. The number of floating point operations required for each solver is estimated for different system sizes. Leu's algorithm requires the fewest operations. However, this number is comparable with both Kane and Trevelyan and Wang's methods. Trevelyan *et al.* use the knowledge gained through these comparisons to develop a new, interactive tool for two-dimensional BE re-analysis. Trevelyan and Scales [147] go on to enhance this two-dimensional scheme by employing a complete approximate LU preconditioner to improve convergence of

the GMRES algorithm during re-analysis. The preconditioner used is the LU decomposition of a previous system matrix, $[A_{i-j}]$, where j is the number of iterations carried out since the LU decomposition was last updated. The benefits of applying a preconditioner are discussed in Section 5.4.1.

5.3 Direct solvers

5.3.1 Gauss elimination

Gauss elimination is a method by which a square matrix can be reduced to its upper triangular form. If this is applied to $[A]$ in equation (5.1), a simple back substitution can then be used to solve the system. This makes Gauss elimination a powerful method for solving multiple problems where only $\{b\}$ has changed. This could be used to rapidly re-solve BE problems where the geometry is constant but the boundary conditions are changing.

Applying partial or full pivoting, whereby rows are swapped so that the largest terms appear on the diagonal of the matrix, improves the accuracy of the method by reducing the effects of numerical rounding [148]. The algorithm for the fully pivoted Gauss elimination is given in lines 1-11 of Algorithm 5.1. This replaces $[A]$ with the upper triangular form of the matrix. The rest of the algorithm forms the back substitution which solves the linear system, overwriting $\{b\}$ with the solution $\{x\}$. The algorithm uses $\frac{2}{3}n^3$ operations to carry out the Gauss elimination where n is the size of the system. An additional $\frac{1}{2}n^2$ operations are required for the back substitution.

Algorithm 5.1 Fully pivoted Gauss elimination and back substitution

1: **for** $i = 1, 2, 3, ..., n-1$ **do**
2: Find row ID, k, of maximum term in column i of A, $k \geq i$
3: Swap rows i and k in A and b
4: **for** $j = i+1, i+2, i+3, ..., n$ **do**
5: $p = A_{ji}/A_{ii}$
6: **for** $k = i, i+1, i+2, ..., n$ **do**
7: $A_{jk} = A_{jk} - pA_{ik}$
8: **end for**
9: $b_j = b_j - pb_i$
10: **end for**
11: **end for**
12: $b_n = b_n/A_{nn}$
13: **for** $i = neq-1, neq-2, neq-3, ..., 1$ **do**
14: **for** $j = i+1, i+2, i+3, ..., n$ **do**
15: $b_i = b_i - A_{ij}b_j$
16: **end for**
17: $b_i = b_i/A_{ii}$
18: **end for**

5.3.2 LU decomposition

LU decomposition is used to factorise a matrix as the product of a pair of matrices:

$$[A] = [L][U] \tag{5.2}$$

where $[L]$ and $[U]$ are of the form:

$$[L] = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{2,1} & 1 & 0 & \cdots & 0 \\ l_{3,1} & l_{3,2} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n,1} & l_{n,2} & l_{n,3} & \cdots & 1 \end{bmatrix} \qquad [U] = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ 0 & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ 0 & 0 & u_{3,3} & \cdots & u_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{n,n} \end{bmatrix} \tag{5.3}$$

To save computer memory, the LU decomposition is commonly stored as a single matrix, overwriting the diagonal terms for $[L]$ with those from $[U]$:

$$[LU] = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ l_{2,1} & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ l_{3,1} & l_{3,2} & u_{3,3} & \cdots & u_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n,1} & l_{n,2} & l_{n,3} & \cdots & u_{n,n} \end{bmatrix} \tag{5.4}$$

The lower triangular matrix, $[L]$ can be constructed by placing the factor, p, into $[L_{ji}]$ each time it is calculated in line 5 of Algorithm 5.1. The upper triangular matrix, $[U]$, is identical to the upper triangle of the modified $[A]$ matrix after Algorithm 5.1 has been run up to line 11.

Once the LU decomposition has been constructed, $[L]$ and $[U]$ can be substituted for $[A]$ in the linear system and the system solved using a forward and backward substitution:

$$[A]\{x\} = [L][U]\{x\} = \{b\} \tag{5.5}$$

$$\text{solve } [L]\{y\} = \{b\} \text{ for } \{y\} \tag{5.6}$$

$$\text{solve } [U]\{x\} = \{y\} \text{ for } \{x\} \tag{5.7}$$

requiring n^2 operations. The process is summarised in Algorithm 5.2 which overwrites $\{b\}$ with the solution $\{x\}$.

Algorithm 5.2 Forward and backward substitution

```
1: for i = 2, 3, 4, ..., n do
2:     for j = 1, 2, 3, ..., i − 1 do
3:         b_i = b_i − LU_ij b_j
4:     end for
5: end for
6: b_n = b_n / LU_nn
7: for i = n − 1, n − 2, n − 3, ..., 1 do
8:     for j = i + 1, 2, 3, ..., n do
9:         b_i = (b_i − b_j LU_ij) / LU_ii
10:    end for
11: end for
```

5.4 Iterative solvers

Iterative solvers provide an efficient method of solving large linear systems. They are also a powerful tool when the system matrix, $[A]$, is changing between analysis runs. They generally work by iteratively

updating an initial approximation, $\{x'\}$, of the solution until the L_2-norm of residual:

$$\{r\} = \{b\} - [A]\{x'\} \tag{5.8}$$

has been reduced to within a specified tolerance. The L_2-norm of $\{r\}$ is defined as:

$$\|\{r\}\| = \sqrt{\sum_{i=1}^{n} r_i^2} \tag{5.9}$$

Many different iterative solvers have been developed, the most common of which are discussed in [149]. The GMRES, BiCGSTAB and TFQMR algorithms have been chosen for this research as they are stable and applicable to non-symmetric systems. They are commonly applied to large sparse matrices typical of the finite element method (FEM). Here they are applied to the smaller fully-populated matrices produced by the BEM. The implementations are based on the templates given in [149] but have been modified to re-use as many vectors as possible to reduce data storage and memory accesses. It can be shown that if the data are spread over a larger area of memory the algorithms run more slowly. To reduce memory requirements, the maximum number of iterations of each algorithm has been limited, enabling a suitable block of memory to be allocated beforehand. If this number of iterations is reached an error message is returned.

Nachtigal *et al.* [150] show that, for any class of problem, there is usually an iterative method which outperforms the other approaches. All of the methods discussed in this section must therefore be assessed with the types of problem encountered in this work to establish the most appropriate for the current application.

5.4.1 Preconditioning

The condition number, κ, of a linear system determines how fast an iterative solver will converge when applied to the linear system given in equation (5.1). The condition number is determined by comparing the maximum and minimum eigenvalues, α, of the system:

$$\kappa = \left| \frac{\alpha_{\max}}{\alpha_{\min}} \right| \tag{5.10}$$

A well-conditioned system has a low condition number, the closer κ is to 1, the better the condition of the system. An ill-conditioned system has a high condition number. If $\kappa = \infty$, the system is singular. Preconditioning may be applied to the linear system with the aim of improving the clustering of the eigenvalues, thereby reducing κ and accelerating convergence of an iterative solver. The most common form of preconditioning, used in the iterative solvers presented in this thesis, is left preconditioning. This pre-multiplies both sides of the linear system by the preconditioner, $[M]$:

$$[M][A]\{x\} = [M]\{b\} \tag{5.11}$$

Left preconditioned iterative solvers aim to minimise the modified residual:

$$\{r'\} = [M](\{b\} - [A]\{x'\}) \tag{5.12}$$

Two forms of preconditioning have been applied to the GMRES, BiCGSTAB and TFQMR algorithms presented in this work. These are Jacobi or diagonal preconditioning and complete approximate LU [17] preconditioning. The applied LU preconditioner is the complete LU decomposition of $[A_0]$, generated

during the initial analysis. When using a simple forward and backward substitution this is equivalent to applying $[M] = [A_0]^{-1}$ to the updated system:

$$[A_0]\{r'\} = [L][U]\{r'\} = \{b\} - [A]\{x'\}(= \{r\}) \tag{5.13}$$

$$\text{solve } [L]\{y\} = \{b\} - [A]\{x'\} \text{ for } \{y\} \tag{5.14}$$

$$\text{solve } [U]\{r'\} = \{y\} \text{ for } \{r'\} \tag{5.15}$$

For clarity the complete approximate LU preconditioning presented here will be referred to simply as LU preconditioning throughout this work.

Diagonal preconditioning is efficient for diagonally dominant matrices, such as those produced by the BEM. It uses the inverse of the diagonal matrix of $[A]$ to construct $[M]$ as used in equation (5.12). This preconditioner can be easily applied by dividing each term in $\{r\}$ by the associated diagonal term in $[A]$:

$$\{r'_k\} = \{r_k\}/[A_{kk}] \tag{5.16}$$

The matrix, $[A_i]$, and the diagonally and LU preconditioned matrices, $[M][A_i]$, $i > 0$, are shown in Figure 5.1, where the colours indicate the value in each cell of the matrix. To minimise the solve time, the preconditioned matrix should be close to the identity matrix and feature well clustered eigenvalues. The BEM produces large terms along the diagonal relative to the rest of the matrix which results in a weak conditioning as shown in Figure 5.1(a). However, this conditioning can be improved through the application of an appropriate preconditioner. As shown in Figure 5.1(c), LU preconditioning produces the best results. It is, however more computationally expensive to apply than diagonal preconditioning, shown in Figure 5.1(b), and is therefore not always be the most suitable option, especially as the LU decomposition of $[A_0]$ may not always be available. It should be noted that the multiplication $[M][A_i]$ is never carried out as it is computationally expensive, it is only used here to show the effect of $[M]$ on $[A_i]$ were it to be applied directly.

A basic method to increase the efficiency of the program is to update the preconditioner in a separate thread. This approach is used by both Gravvanis and Giannoutakis [151] and Courtecuisse *et al.* [152], with the FEM, and Trevelyan and Scales [17], with the BEM. Courtecuisse *et al.* [152] also use a matrix warping technique to modify the preconditioner to extend its usefulness.

If it is assumed that each geometric update is small, $\{x_{i-1}\}$ can be used to reduce the solution time by providing a good first approximation of $\{x_i\}$ [17].

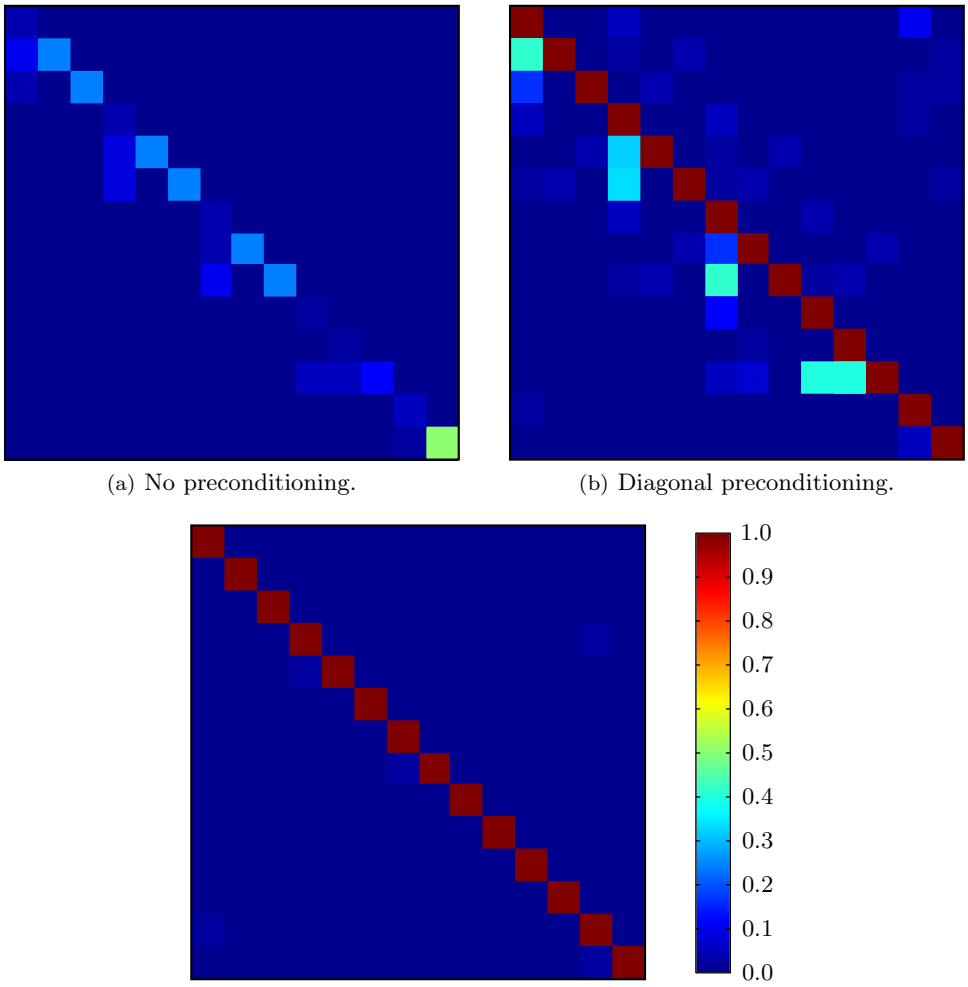
5.4.2 GMRES

The GMRES algorithm extends the minimal residual (MINRES) method to solve non-symmetric systems. It is a Krylov subspace method. The basis of the Krylov subspace is that the inverse of a matrix can be found in terms of a linear combination of its powers. In this case, GMRES uses the Arnoldi method to generate a series of orthogonal vectors, $\{\phi_j\}$ (where j is the iteration number), in the Krylov subspace that can be used as a basis in which to construct the solution vector. Together with factors ζ_j, these are used to compute the solution:

$$\{x_i\} = \{x_{i-1}\} + \zeta_1\{\phi_1\} + \zeta_2\{\phi_2\} + ... + \zeta_j\{\phi_j\} \tag{5.17}$$

or

$$\{x_i\} = \{x_{i-1}\} + [\Phi]\{\zeta\} \tag{5.18}$$



(a) No preconditioning.

(b) Diagonal preconditioning.

(c) LU preconditioning.

Figure 5.1: Effect of applying preconditioning to a matrix.

Factor ζ_j is chosen to minimise the residual, $\{r_j\}$, where $\{r_j\} = \{b_i\} - [A_i]\{x_{ij}\}$. Convergence is reached when $\|\{r_j\}\|/\|\{b_i\}\| < tol$ where $\|\cdot\|$ denotes the L_2-norm. The tolerance, tol, is defined by the user and is commonly 10^{-6}.

A new basis vector is generated with each iteration of the algorithm. Once the method has converged, equation (5.18) is applied to update $\{x_i\}$. The amount of memory required by the solver, in addition to that required to store the system and preconditioner, and the time required to compute each new Krylov vector increases with each iteration. To limit these requirements, restarted versions of the GMRES method are often used, updating $\{x_i\}$ before each restart. In the worst case, GMRES will always converge to the correct solution in n steps. However, to run the algorithm for this many iterations would be significantly less efficient than applying a direct solver.

A flowchart of the non-restarted GMRES algorithm is given in Figure 5.2.

5.4.3 BiCGSTAB

The BiCGSTAB method was developed as an improvement on the conjugate gradient squared (CGS) method that often exhibits irregular convergence patterns.

$$\{r_j\} = Q_j([A_i])P_j([A_i])\{r_0\} \tag{5.19}$$

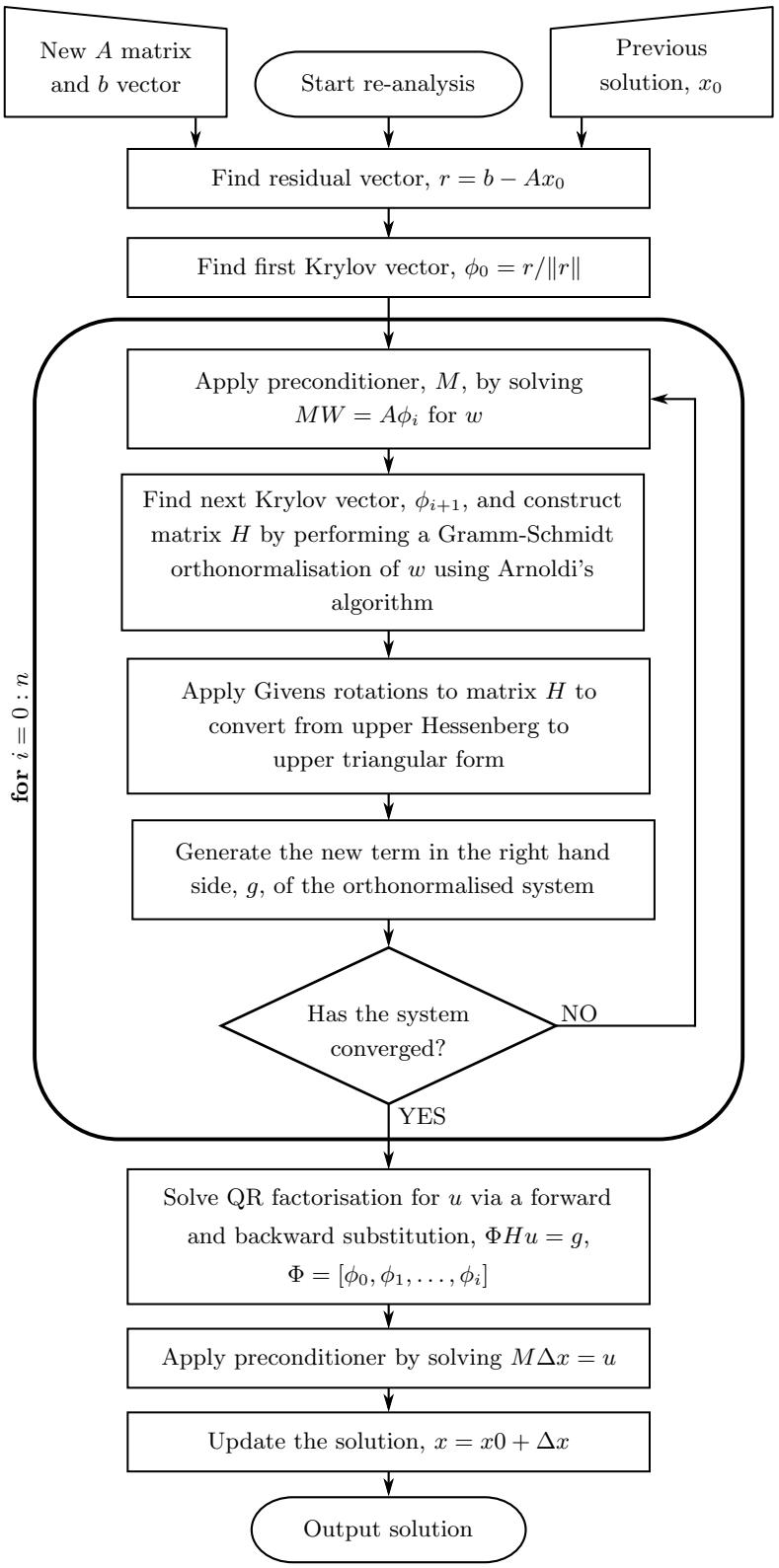


Figure 5.2: GMRES flowchart.

where j is the iteration number, $P_j([A_i])$ is a jth degree polynomial in $[A_i]$ and $Q_j([A_i])$ is a jth degree polynomial describing a steepest descent update. The vector $\{x_i\}$ is updated and convergence checked, using the same stopping criterion as the GMRES method, twice in each iteration. The BiCGSTAB method requires about half as much memory as the GMRES method at $6n$ values.

A flowchart of the BiCGSTAB algorithm is given in Figure 5.3.

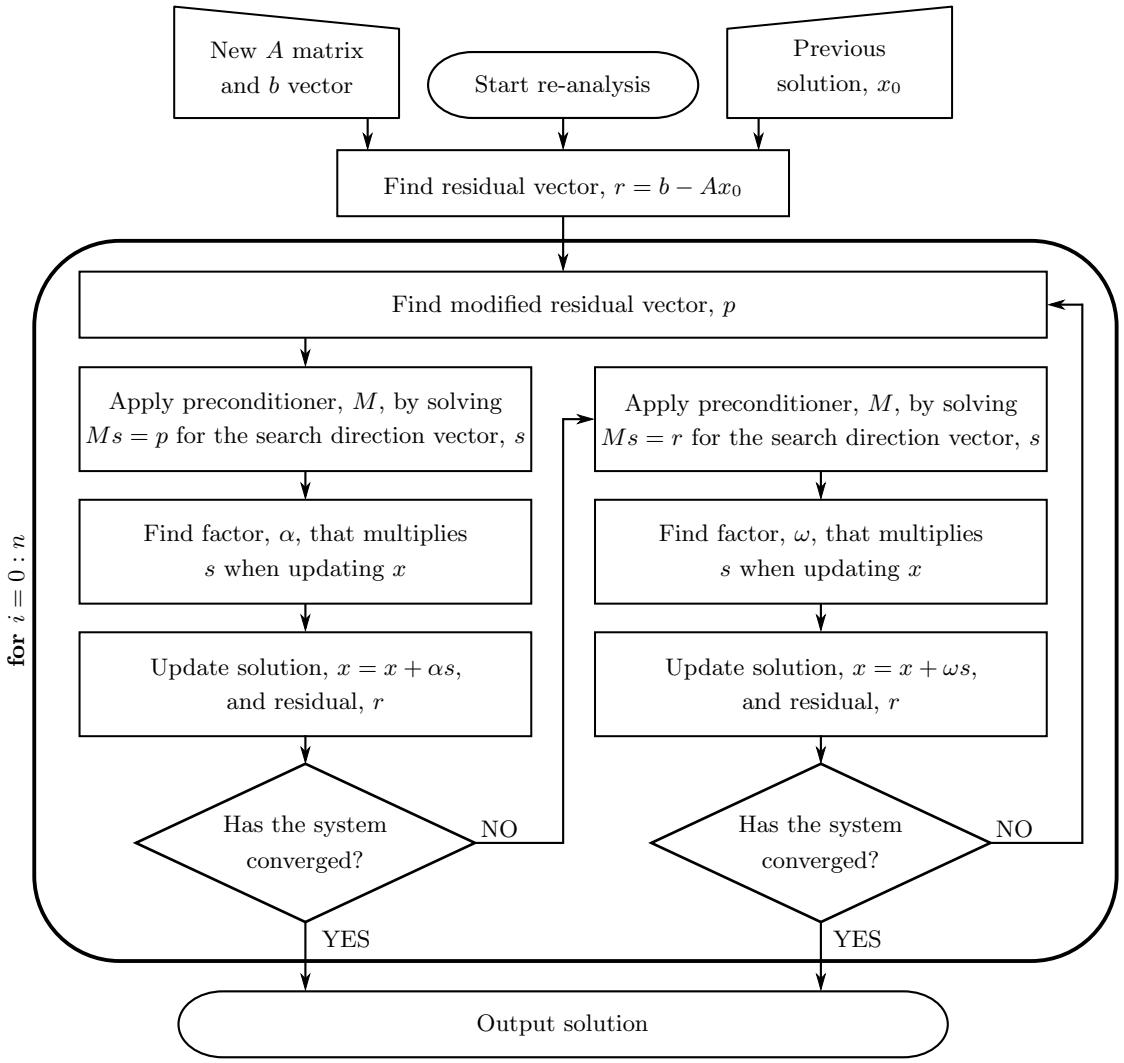


Figure 5.3: BiCGSTAB flowchart.

5.4.4 TFQMR

The quasi-minimal residual (QMR) method [153] aims to solve the system in a least squares sense, in a similar manner to the GMRES approach. However, the basis generated in the Krylov subspace is bi-orthogonal and the residual is therefore not a true minimisation, instead being referred to as quasi-minimal. The TFQMR [140] algorithm achieves this without using $[A]^T$ which is slow to access due to the way a matrix is stored in computer memory. The total additional memory required by the TFQMR algorithm is similar to that of the BiCGSTAB algorithm at $7n$ values.

The basic TFQMR algorithm is shown in Figure 5.4. Each iteration adds a new vector to one of the two bi-orthogonal systems. For implementation the algorithm has has been expanded to reduce computation. This means that each loop around the algorithm is effectively two iterations of the flowchart in Figure 5.4.

The vector $\{x_i\}$ is updated after each of these iterations, therefore only a single basis vector is required at any one time, thus reducing the memory requirements. The stopping condition computes an upper bound for the residual, $\{r_j\}$, using data already stored by the algorithm. This is compared to *tol*.

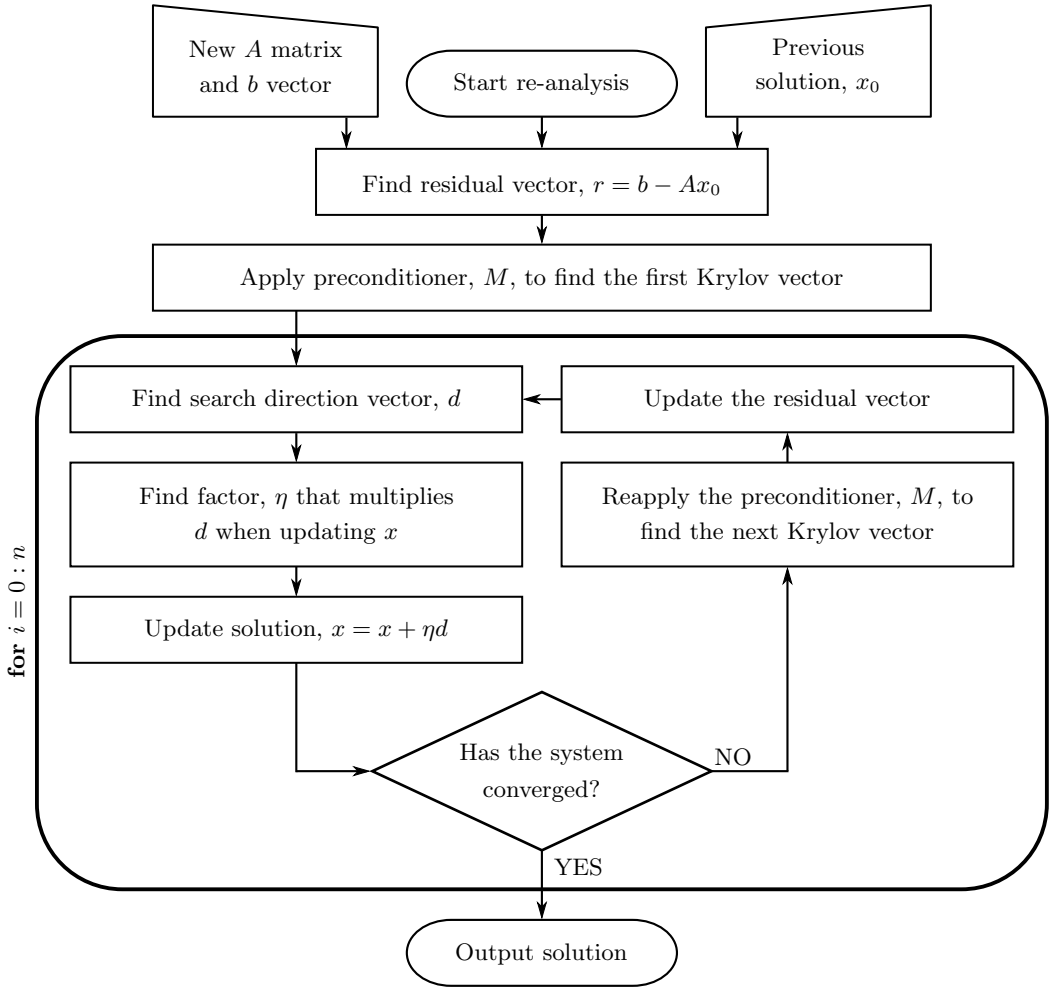


Figure 5.4: TFQMR flowchart.

5.4.5 Acceleration of the iterative solvers using adaptive cross approximation

Any admissible block, $[A']$, can be approximated through a combination of s vectors:

$$[A'] \approx [S] = [U][V]^T \tag{5.20}$$

where:

$$[U] = [\{u_1\}, \{u_2\}, ..., \{u_s\}] \tag{5.21}$$

$$[V] = [\{v_1\}, \{v_2\}, ..., \{v_s\}] \tag{5.22}$$

These vectors can be constructed through adaptive cross approximation (ACA) as described in Section 4.3.

By using this approximation, the number of operations required during matrix-vector multiplication, such as is commonly employed in iterative solvers, can be reduced. To compute the product, $[A']\{b'\}$, n^2 operations would normally be required, where an operation consists of a multiplication and an addition.

If this were re-written:

$$[A']\{b'\} \approx [S]\{b'\} = [U]\left([V]^T\{b'\}\right) \tag{5.23}$$

only $2ns$ operations would be required.

Using this approach, efficiency gains can be made in the theoretical solution of equation (5.1) if $s < n/2$. For small systems it is expected that $s \to n$ as subdivision of the problem will result in the majority of blocks being small. However, for much larger systems more large blocks will exist where $s \ll n$, thus providing increased acceleration of the solution.

5.5 Reduction techniques

Reduction techniques reduce the size of the problem by approximating it with a smaller system. This reduced system is very fast to solve but additional overheads are required to generate the system. If a set of m basis vectors, $[\Phi]$, together with m coefficients, $\{\zeta\}$, are defined, $\{x_i\}$ can be written in the form:

$$\{x_i\} = [\Phi]\{\zeta\} \tag{5.24}$$

By substituting (5.24) into (5.1) and pre-multiplying by $[\Phi]^T$ the reduced system can be produced:

$$[\Phi]^T[A_i][\Phi]\{\zeta\} = [\Phi]^T\{b_i\} \tag{5.25}$$

The basis, $[\Phi]$, is derived from some representative solutions of the initial problem and is of size $n \times m$, so it remains only to solve a small $m \times m$ system where $m \ll n$. Hence $\{\zeta\}$ can be found with a direct solver.

5.5.1 Leu's reduction method

Leu [23] presents a reduction method, formulated specifically for the BEM, to solve equation (5.1), which extends Kirsch's [154] reduction method to the BEM and applies it to re-analysis. Leu uses a Gram-Schmidt orthonormalisation procedure to modify the basis vectors, φ_i, resulting in an uncoupling of the reduced system. These basis vectors are created iteratively and a convergence criterion is used to determine the number required for a specified level of accuracy. The solution, $\{x_i\}$, is updated as the basis is constructed.

The method requires the inverse of the initial system matrix, $[A_0]^{-1}$. This can be achieved by applying the LU decomposition by means of a forward and backward substitution. Leu assumes that each re-analysis is applied to a different perturbation of the initial model rather than an iterative update of the last model solved. It can be shown that Leu's algorithm requires fewer operations for the solution phase of re-analysis than Kane [3].

To construct the basis, Leu first pre-multiplies (5.1) by $[A_0]^{-1}$:

$$([I] + [\hat{A}])\{x_i\} = \{\hat{b}\} \tag{5.26}$$

where:

$$[\hat{A}] = [A_0]^{-1}([A_i] - [A_0]) \tag{5.27}$$

$$\{\hat{b}\} = [A_0]^{-1}\{b_i\} \tag{5.28}$$

Equation (5.26) can be rearranged and expanded such that:

$$\{x\} = ([I] + [\hat{A}])^{-1}\{\hat{b}\} = ([I] + [\hat{A}] + [\hat{A}]^2 - [\hat{A}]^3 + \ldots - [\hat{A}]^n)\{\hat{b}\} \tag{5.29}$$

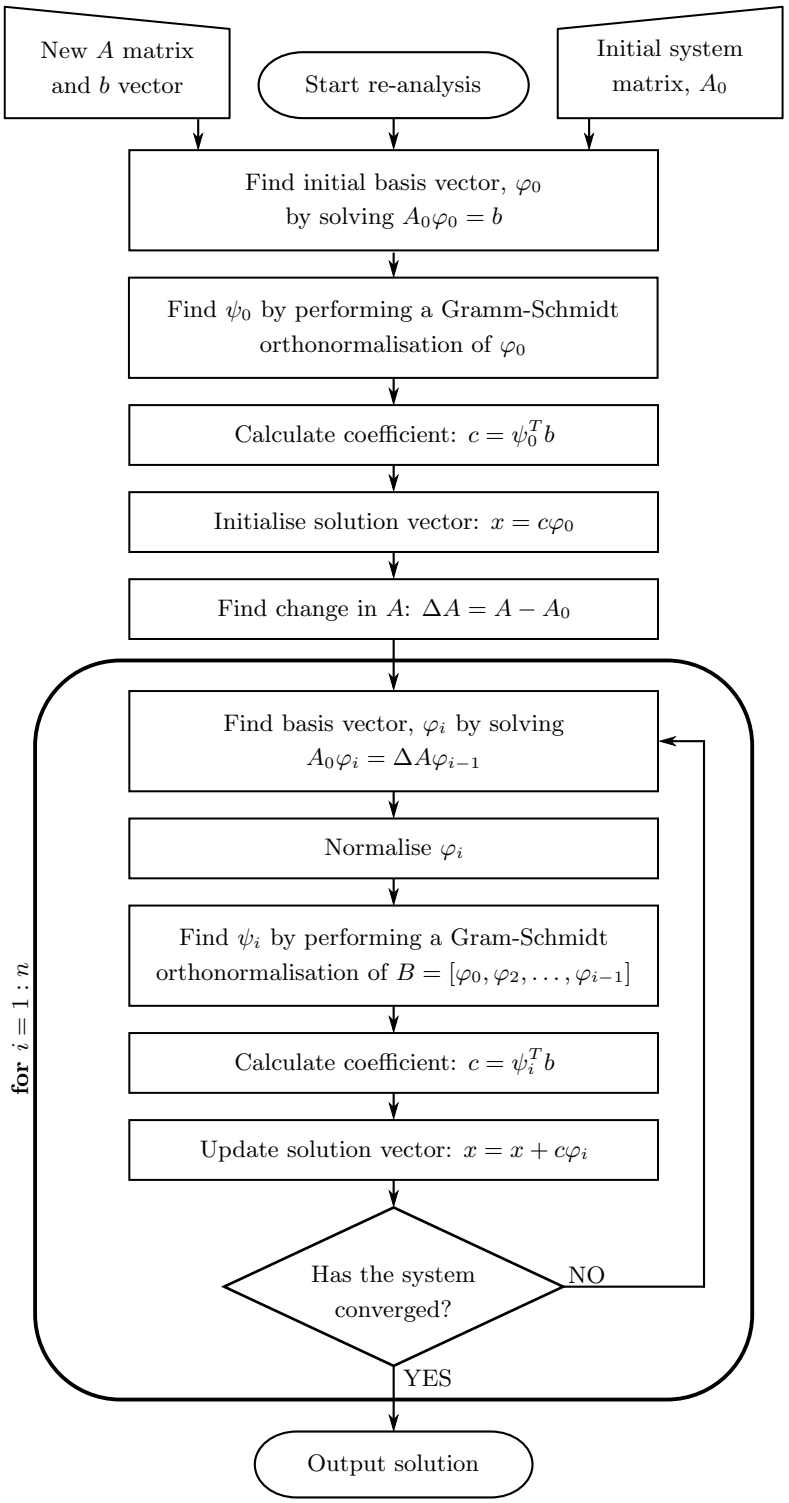


Figure 5.5: Leu flowchart.

The first m terms in this sequence can be used to construct a basis, $[\Phi]$:

$$[\Phi] = \Big[\{\varphi_0\} \quad \{\varphi_1\} \quad \{\varphi_2\} \quad \cdots \quad \{\varphi_m\}\Big] = \Big[\{\hat{b}\} \quad [\hat{A}]\{\hat{b}\} \quad [\hat{A}]^2\{\hat{b}\} \quad \cdots \quad [\hat{A}]^{m-1}\{\hat{b}\}\Big] \tag{5.30}$$

The basis vectors can be iteratively calculated if it is realised that:

$$\{\varphi_i\} = [\hat{A}]\{\varphi_{i-1}\} \tag{5.31}$$

hence:

$$[A_0]\{\varphi_i\} = ([A_i] - [A_0])\{\varphi_{i-1}\} \tag{5.32}$$

To uncouple the reduced system a Gram-Schmidt orthonormalisation is applied to equation (5.25). This produces a second set of basis vectors, $[\Psi] = \Big[\{\psi_0\} \quad \{\psi_1\} \quad \{\psi_2\} \quad \cdots \quad \{\psi_m\}\Big]$, such that:

$$[\Psi]^T[A_i][\Phi] = [I] \tag{5.33}$$

The new basis can be used to generate the coefficients, ζ:

$$\{\zeta\} = [\Psi]^T\{b_i\} \tag{5.34}$$

which can also be used as an error measure to assess convergence:

$$tol > \frac{\zeta_i}{\sum_{j=0}^{i}\zeta_j} \tag{5.35}$$

where i is a count of the number of iterations of the algorithm, which is the same as the current number of basis vectors.

Leu's algorithm is summarised in Figure 5.5.

5.5.2 Proper orthogonal decomposition

Proper orthogonal decomposition (POD) (also called *Karhunen-Loève decomposition (KLD)*) is a key part of reduced order modelling (ROM). A good review of recent advances in ROM is given by Xiao *et al.* [155]. Different approaches may be adopted for generating a suitable basis for POD; two are presented here. Ryckelynck *et al.* [25] formulate a solution specifically for the BEM (a similar formulation for the FEM can be found in [156]). The vectors $\{x_i\}$ from the first s analysis runs are used to construct matrix $[Q]$, which is used to produce matrix $[K]$:

$$[Q] = \Big[\{x_0\} \quad \{x_1\} \quad \cdots \quad \{x_s\}\Big] \tag{5.36}$$

$$[K] = [Q][Q]^T \tag{5.37}$$

from which m eigenvectors are selected to form the basis, $[\Phi]$, based on the related eigenvalues, α_k, where $\alpha_k > 10^{-10}\alpha_{\max},\ (k = 1, 2, 3, ..., n)$ and $\alpha_{\max}$ is the highest eigenvalue. In the current work, this method has been denoted E-POD.

Kerfriden *et al.* [26] formulate their method for the FEM. They use a set of s vectors made up from a representative family of solutions to the initial model under differing initial conditions. These vectors are combined as in equation (5.36) to form $[Q]$ and the singular value decomposition (SVD) is found:

$$[Q] = [U][S][V]^T \tag{5.38}$$

The first m columns of $[U]$ are taken to form an orthonormal basis $[\Phi]$ where $m < s$. This method has herein been denoted SVD-POD.

Once $[\Phi]$ has been constructed it can be used in conjunction with (5.25) to find $\{\zeta\}$ and hence $\{x_i\}$ from (5.24), as shown in Figure 5.6.

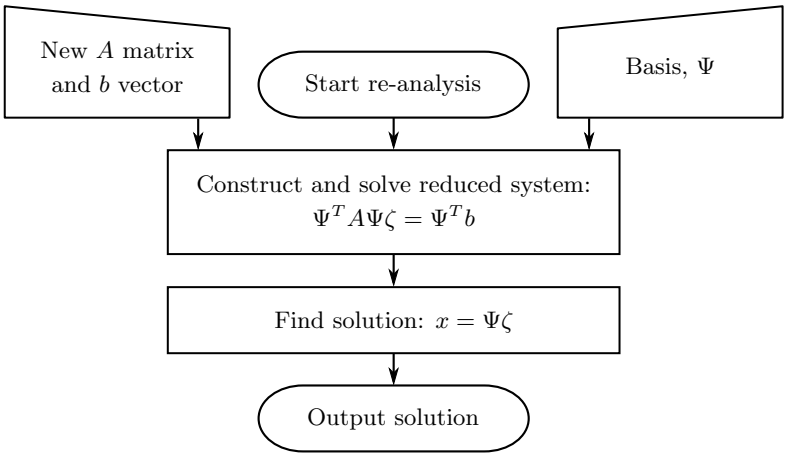


Figure 5.6: POD flowchart.

5.5.3 Static condensation

An alternative approach is *static condensation*. This aims to reduce solve time by only re-solving the parts of the linear system that have been updated. The new system is subdivided into blocks based on the updated rows and columns:

$$\begin{bmatrix} [A_{11}] & [A_{12}] \\ [A_{21}] & [A_{22}] \end{bmatrix} \begin{Bmatrix} \{x_1\} \\ \{x_2\} \end{Bmatrix} = \begin{Bmatrix} \{b_1\} \\ \{b_2\} \end{Bmatrix} \tag{5.39}$$

where $[A_{11}]$ contains terms that have not been modified. By multiplying out the first line of equation (5.39):

$$\{x_1\} = [A_{11}]^{-1} \{\{b_1\} - [A_{12}]\{x_2\}\} \tag{5.40}$$

and substituting into the second line, a reduced system can be constructed:

$$[S]\{x_2\} = \big\{\{b_2\} - [A_{12}][A_{11}]^{-1}\{b_1\}\big\} \tag{5.41}$$

where $[S]$ is the Schur complement:

$$[S] = \big[[A_{22}] - [A_{21}][A_{11}]^{-1}[A_{12}]\big] \tag{5.42}$$

The system can now be solved to find $\{x_2\}$. The remainder of the solution vector, $\{x_1\}$, can be found by substituting $\{x_2\}$ into equation (5.40). This method could easily be applied where new rows and columns are added to the system as they could simply be appended to $[A_{12}]$, $[A_{21}]$ and $[A_{22}]$.

If $[A_{11}]^{-1}$ is known *a priori*, the approximate number of operations (one multiplication and one

addition) can be summarised as:

$$\left[[A_{22}] - [A_{21}][A_{11}]^{-1}[A_{12}]\right] \Rightarrow n^2 n_u - n n_u^2 \tag{5.43}$$

$$\left\{\{b_2\} - [A_{12}][A_{11}]^{-1}\{b_1\}\right\} \Rightarrow 2(n-n_u)^2 \tag{5.44}$$

$$\text{solve} \Rightarrow 2n_u^3/3 \tag{5.45}$$

$$[A_{11}]^{-1}\left\{\{b_1\} - [A_{12}]\{x_2\}\right\} \Rightarrow 2(n-n_u)^2 \tag{5.46}$$

$$O_{SC} = 2n_u^3/3 - n_u^2(n+4) + n_u(n^2-8n) + 4n^2 \tag{5.47}$$

where n is the original size of the system and n_u is the number of updated rows and columns.

If O_{SC} is equated to the number of operations required by a Gauss elimination ($O_G = 2n^3/3$), the following equation is produced:

$$2n_u^3/3 - n_u^2(n+4) + n_u(n^2-8n) - 2n^3/3 + 4n^2 = 0 \tag{5.48}$$

This can be solved for a range of values of n to show that static condensation will always solve the system faster than a direct solver. However, care should be taken when interpreting this result as it assumes that $[A_{11}]^{-1}$ is known *a priori* and is the maximum possible size for the modified problem; ie. all the terms in $[A_{11}]^{-1}$ will never change and all the terms not in $[A_{11}]^{-1}$ will be updated. The saving in solve time when using static condensation is shown in Figure 5.7.

If a similar comparison is carried out comparing the number of operations to a GMRES solver, the formula for the size of system where static condensation should be used instead of GMRES can be approximated as:

$$n < 3.8s(n_u/n)^{-0.8} + 20(n_u/n) \tag{5.49}$$

where s is the number of iterations in the GMRES solver. Typically $s < 20$ and $n_u/n > 0.2$ for the type of problems encountered in this work. Under these conditions static condensation cannot improve on the GMRES algorithm, as shown in Figure 5.7.

For maximum efficiency when applying static condensation, exactly which rows and columns will be updated must be known *a priori* so that $[A_{11}]^{-1}$ can be constructed. As this is unknown, $[A_{11}]$ must be smaller than the optimum size to avoid including any rows or columns that could change. Additional efficiency losses will be incurred as a result.

5.5.4 Woodbury

The *Woodbury formula* [157] and *Sherman-Morrison formula* [158] (a special case of the Woodbury formula) update the inverse of a matrix when n_u rows and columns have been modified. Wang *et al.* [10] use this method for real-time analysis to add rows and columns to an otherwise unchanged matrix.

As with static condensation, the matrix is subdivided into four parts:

$$[A] = \begin{bmatrix} [A_{11}] & [A_{12}] \\ [A_{21}] & [A_{22}] \end{bmatrix} \tag{5.50}$$

where $[A_{11}]$ is the unchanged part of the matrix. The inverse of the updated matrix is given as:

$$[A]^{-1} = \begin{bmatrix} [\tilde{A}_{11}] & [\tilde{A}_{12}] \\ [\tilde{A}_{21}] & [\tilde{A}_{22}] \end{bmatrix} \tag{5.51}$$

and can be constructed using the following set of equations:

$$[\tilde{A}_{11}] = [A_{11}]^{-1} + [A_{11}]^{-1}[A_{12}][S]^{-1}[A_{21}][A_{11}]^{-1} \tag{5.52}$$

$$[\tilde{A}_{12}] = -[A_{11}]^{-1}[A_{12}][S]^{-1} \tag{5.53}$$

$$[\tilde{A}_{21}] = -[S]^{-1}[A_{21}][A_{11}]^{-1} \tag{5.54}$$

$$[\tilde{A}_{22}] = [S]^{-1} \tag{5.55}$$

where $[S]$ is the Schur complement as given in equation (5.42).

If it is assumed that $[A_{11}]^{-1}$ is known *a priori*, the approximate number of operations involved in completing the linear solve can be summarised as:

$$[S]^{-1} \Rightarrow n^2 n_u + n_u^2 n + n_u^3 \tag{5.56}$$

$$[\tilde{A}_{11}] \Rightarrow n^2 n_u + 2n_u^2 n \tag{5.57}$$

$$[\tilde{A}_{12}] \Rightarrow n_u^2 n \tag{5.58}$$

$$[\tilde{A}_{21}] \Rightarrow n_u^2 n \tag{5.59}$$

$$[\tilde{A}_{22}] \Rightarrow 0 \tag{5.60}$$

$$[A]^{-1}\{b\} \Rightarrow n^2 \tag{5.61}$$

$$O_W = n_u^3 + 5n_u^2 n + 2n_u n^2 + n^2 \tag{5.62}$$

If $n_u \ll n$ then this can be approximated as $2n^2 n_u$. By equating the number of operations to those in a direct solver (such as a Gauss elimination) the following formula is constructed:

$$n_u^3 + 5n_u^2 n + 2n_u n^2 - 2n^3/3 + n^2 = 0 \tag{5.63}$$

For efficiency gains to be made for all values of n, $n/n_u < 0.215$. Further analysis indicates that the Woodbury formula can only outperform static condensation when $n_u/n < 0.003$. This is clearly shown in Figure 5.7.

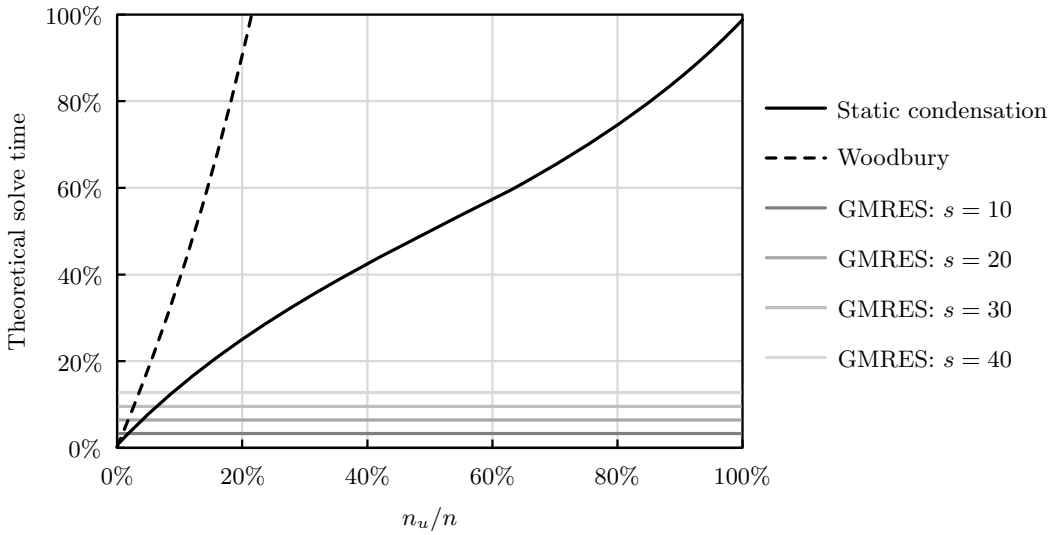


Figure 5.7: Theoretical speed-up over a direct solver.

5.6 Frontal solvers

Frontal solvers were first proposed by Irons in 1970 [159]. These solve the linear system as it is being constructed, using a similar technique to Gauss elimination. This is particularly useful for solving sparse systems, typically found in the FEM, as the solver can avoid carrying out operations that multiply zero terms. The full linear system is never created explicitly, thereby reducing the memory requirements of the algorithm. Only the front, which contains the terms currently being assembled is stored at any one time. The dense matrix operations involved in processing the front use the CPU efficiently. The method can be parallelised by implementing a multi-frontal solver, whereby several independent fronts can be assembled on different processors [160].

Frontal solvers are not as efficient for solving dense matrices and, as parts of the linear system of equations are re-used to accelerate the re-analysis algorithms presented in this thesis, the system must be created in full. This negates the major benefits of applying a frontal solver in this instance and they will therefore not be considered further.

5.7 Parallelising the solvers

The solvers may be parallelised to make use of multiple computer processors and hence accelerate convergence. This parallel code may be customised to run on multiple CPUs, which use a few powerful processors, or graphics processing units (GPUs) which contain hundreds of small processors and are most appropriate for massively parallel problems.

The direct solvers discussed in this work cannot be easily parallelised due to the full pivoting and the substitution routines. The iterative solvers have similar problems in that each iteration must be completed before the next can proceed. For these reasons it is only possible to parallelise operations carried out within each loop of both the direct and iterative solvers discussed in this work. Kreienmeyer and Stein [56] compare the Gauss elimination, GMRES and BiCGSTAB solvers when solving a BEM system on multiple CPUs and concludes that the BiCGSTAB algorithm can be parallelised more efficiently but that the GMRES algorithm outperforms it due to a faster convergence rate.

The forward and backward substitution used to apply the LU decomposition either as a preconditioner or to solve the system cannot easily be parallelised as all the previous solutions are required to compute the next. However, any operations within each iteration of the algorithms can be easily parallelised, thereby reducing the time for the costly matrix-vector multiplications inherent in all of the solution algorithms.

When parallelising the reduction algorithms, all of the operations used in POD may be parallelised as they are simple matrix multiplications. However, Leu's algorithm iteratively creates basis vectors and requires the previous solution to create the next vector and can therefore only be parallelised within each iteration.

More extensively parallel implementations of all these algorithms exist (an overview of packages that implement parallel iterative solvers can be found in [161]) however these often incur large overheads in dividing up the problem. On the GPU, libraries such as Nvidia's CUDA basic linear algebra subprograms (CUBLAS) [162] include functionality to carry out forward and backward multiplication in parallel as well as more simple operations. Couturier and Domas [163] compare the speed of running a standard GMRES solver on both CPUs and GPUs and show a speed-up ranging from eight to twenty-three times for a variety of different FE problems. However, for all these problems n is greater than 147,000 and the system cannot therefore be solved in real-time using current hardware. It is not clear whether the speed-up will scale to small problems that are amenable to real-time analysis. Several strategies for applying direct solvers to dense matrices on the GPU are discussed by Galoppo *et al.* [164]. Whilst these methods

have been applied to fluid flow simulation, they are equally applicable to the BEM. All the strategies show good speed-up over multiple CPUs.

Chapter 6

Initial meshing strategy

"Science never solves a problem without creating ten more."

George Bernard Shaw

6.1 Introduction

A model built by the user must be discretised into a number of elements, forming a surface mesh, before it can be numerically integrated using the boundary integral equation (BIE) discussed in Chapter 2. This necessitates the development of an efficient mesh generating algorithm. The meshing strategy proposed in this chapter produces a high quality mesh and takes into account the need for rapid initial mesh generation. The initial mesh must be of sufficiently high quality so that the perturbations applied to the mesh during the re-meshing procedure do not significantly affect it. A comprehensive data structure will also aid rapid analysis and re-meshing of the model. The principal focus of the present work is the analysis of small mechanical components. These models will best lend themselves to real-time analysis. As these are simple models, only basic geometric features are encountered by the algorithm. Each of these features can therefore be stored as a basic mathematical entity. This creates a highly efficient data storage structure with details about each entity stored in a tree structure.

Existing meshing packages such as *Gmsh* [165] could be used to provide a suitable meshing algorithm for any model. However, this would require in depth interaction with the source code of the packaged mesh generator to extract the data required for re-meshing the model. Additional overheads and data, unnecessary for this work, would also be incurred, thus slowing the algorithm. In order that absolute control can be maintained over the mesh and data structure, the mesh generator used in this work has been entirely written by the author.

For this work, the *Open Cascade* [166] modelling libraries have been utilised to create the modelling environment in which the user can construct and analyse three-dimensional components. These libraries produce some of the data required by the meshing algorithm. However, only enough data for visualisation of the geometry is created. This must be parsed to ensure it is in the correct format and several additional fields calculated.

6.2 The model data structure

6.2.1 Introduction

The algorithms developed for this thesis are coded in an object oriented fashion. A series of classes store data defining various geometric entities. These classes will be filled automatically via the Open Cascade modelling library, which is used to interpret the user input. There are three classes used to define the

geometry of a three-dimensional model. These store vertices, lines and patches. A typical layout of these features is shown in Figure 6.1.

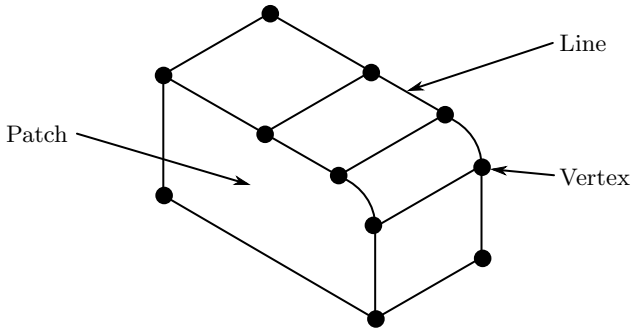


Figure 6.1: Typical basic model.

A vertex is defined as a point in three-dimensional space; a line links two vertices and generally forms an edge of the model. A patch is a surface enclosed by a set of lines, generally forming a face of the model. Each continuous set of lines around a patch are grouped to form a wire, for example, in Figure 6.1 lines 0, 2 and 3 form a wire around patch 0. Note that a vertex must be defined at any point where multiple lines meet, even if there is no change of angle between the lines. This data structure can be summarised for a basic model in a tree, as shown in Figure 6.2. The tree can be easily traversed using information stored within each entity to establish its relationship with any other entity.

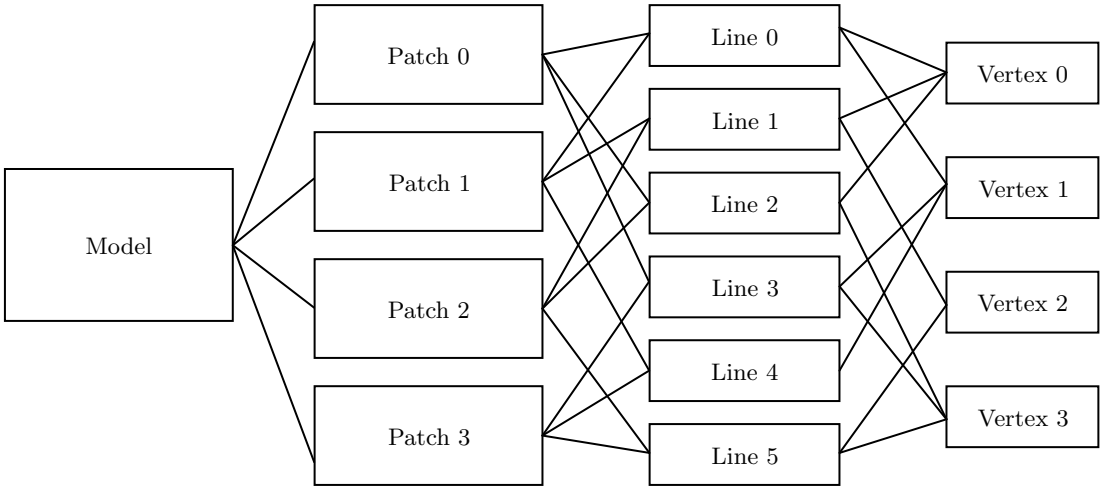


Figure 6.2: Model data structure.

6.2.2 Vertices

The fields in the *vertex class*, which define the location and connectivity, are shown in Table 6.1.

Table 6.1: Vertex class.

Field	**Description**
`float coords[3]`	(x, y, z) coordinates of vertex.
`int numLines`	Number of lines around vertex.
`int lineID[]`	IDs of lines around vertex.
`int end[]`	ID of end of line at vertex.

6.2.3 Lines

The fields stored in the *line class* are shown in Table 6.2. Straight lines are defined by the two vertices at their ends. Curved lines (circular arcs) are defined by the two vertices at their ends, the radius and centre point of the arc. The angle through which the arc turns and the axis of rotation are also included for convenience. A flag defines the type of line. This informs the algorithm to ignore the extra values used to define a curved line if it is dealing with a straight line. Both line types store a reference number for the patches that meet at the line. Fields to store the number of elements along each line and the grading factor between elements are initialised. Additional fields are required if the line is split to allow the segments to grade in or out in the centre of the line. For clarity these are not included in Table 6.2.

Table 6.2: Line class.

Field	**Description**
`int lineType`	Flag defining line type (0: straight; 1: circular arc).
`float length`	Length, l, of line.
`float radius`	Radius, r, of arc.
`double dtheta`	Subtended angle, θ.
`float centreOfCurvature[3]`	(x, y, z) coordinates of centre of arc, C.
`float axisOfRotation[3]`	Unit vector, axis of arc rotation.
`int convex`	Flag line form (0: concave, 1: convex).
`int vertexID[2]`	Start and end vertices of line, V_1, V_2.
`float endElementLength[2]`	Start and end element lengths, δ_1, δ_2.
`float gradingFactor`	Grading factor, λ.
`int numElements`	Number of segments along line.

6.2.4 Patches

The data stored in the *patch class* are shown in Table 6.3. Lines are defined in an anticlockwise direction around the patch boundary when viewed from the outside of the model, as shown in Figure 6.3. Each continuous group of lines forms a wire of which there may be multiple associated with each patch. Mathematically the orientation of the boundary is defined such that the cross product of the outward normal to the patch and the boundary tangent vector (in the same direction as the boundary curve) is always pointing towards the interior of the surface. Flags determine whether the line is traversed from V_1 to V_2 or from V_2 to V_1 and whether the line is on the outer boundary or part of a hole in the patch. The type and form of the patch are also flagged where appropriate.

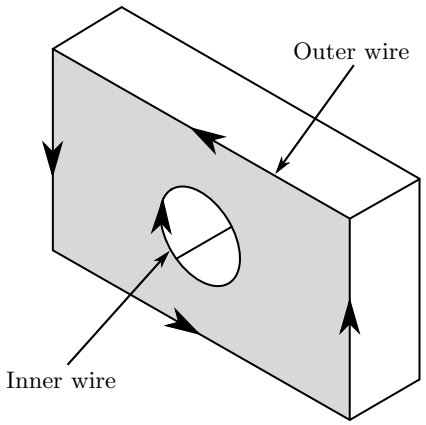


Figure 6.3: Line orientation.

Table 6.3: Patch class.

Field	Description
`int numLines`	Number of lines around patch.
`int lineID[]`	IDs of lines surrounding patch.
`int lineDirectionCorrect[]`	Flag defining line direction (0: start at V_2; 1: start at V_1).
`int numWires`	Number of complete loops of lines that form the boundary of the patch.
`int wire[]`	Flag line location (0: line is around outside of patch; 1+: line is on an internal feature).
`int patchType`	Flag patch type (0: plane; 1: cylindrical; 2: conical; 3: spherical).
`int convex`	Flag patch form (0: concave, 1: convex).
`int normal`	Outward normal of plane patches, N.
`int bcType[3]`	Type and (x, y, z) direction of boundary condition applied to patch (0: traction; 1: displacement).
`float bcValue[3]`	Value of boundary condition, (x, y, z) components.

A single line or patch may form a continuous loop, such as a circle or the surface of a cylinder, so long as they are still fully defined. A vertex must be defined somewhere around a circular line and it should appear twice in the `vertexID` field. The line direction must be defined to match the axis of rotation, obeying the right hand rule. Similarly, a line must be defined to split a continuous patch. This will appear twice in the `lineID` field as it is traversed in opposite directions.

6.2.5 Parsing the data structure

Open Cascade does not store the model data in the structure required for the meshing algorithms developed as part of this work. To overcome this interfacing problem, and to calculate the additional data required, a model parsing algorithm has been developed.

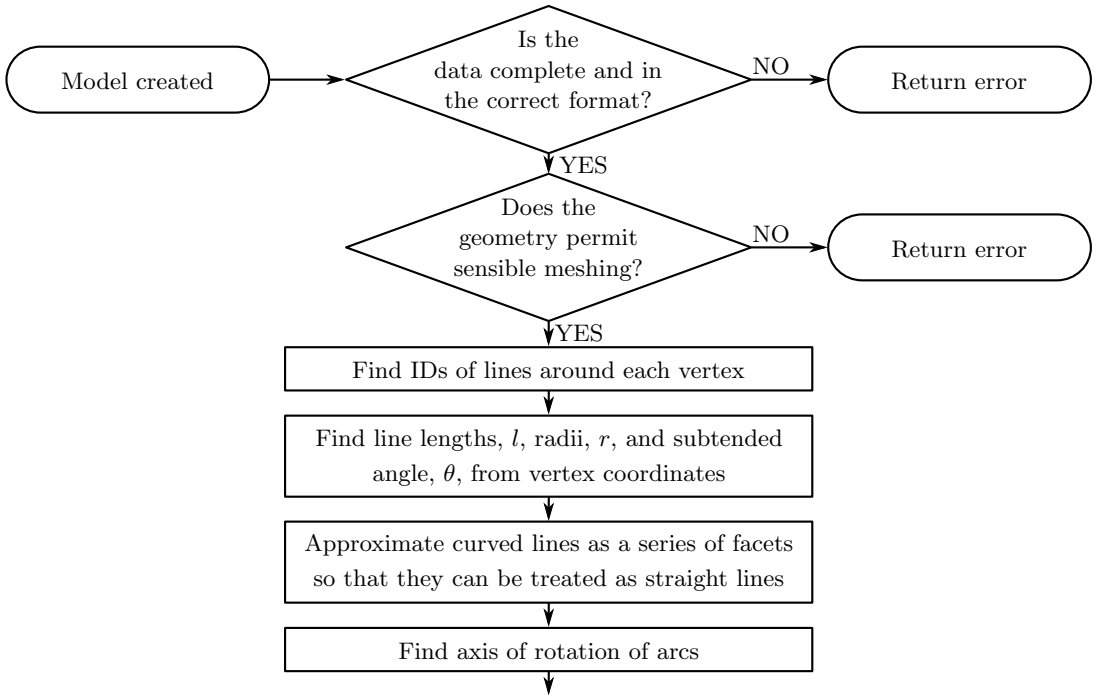


Figure 6.4: Flowchart showing initial checks on the model.

The parsing algorithm can be broadly divided into three parts. The first part, shown in Figure 6.4, checks that all the required data exists and that it is viable to create a mesh from this data. It then generates any extra data that can be easily calculated, such as the line lengths. The second part, shown in Figure 6.5, orders and orientates the direction flags of all the lines so that they are consistent around all patches. The final part, shown in Figure 6.6, checks if these orientations are consistently right or consistently wrong, in which case it inverts them.

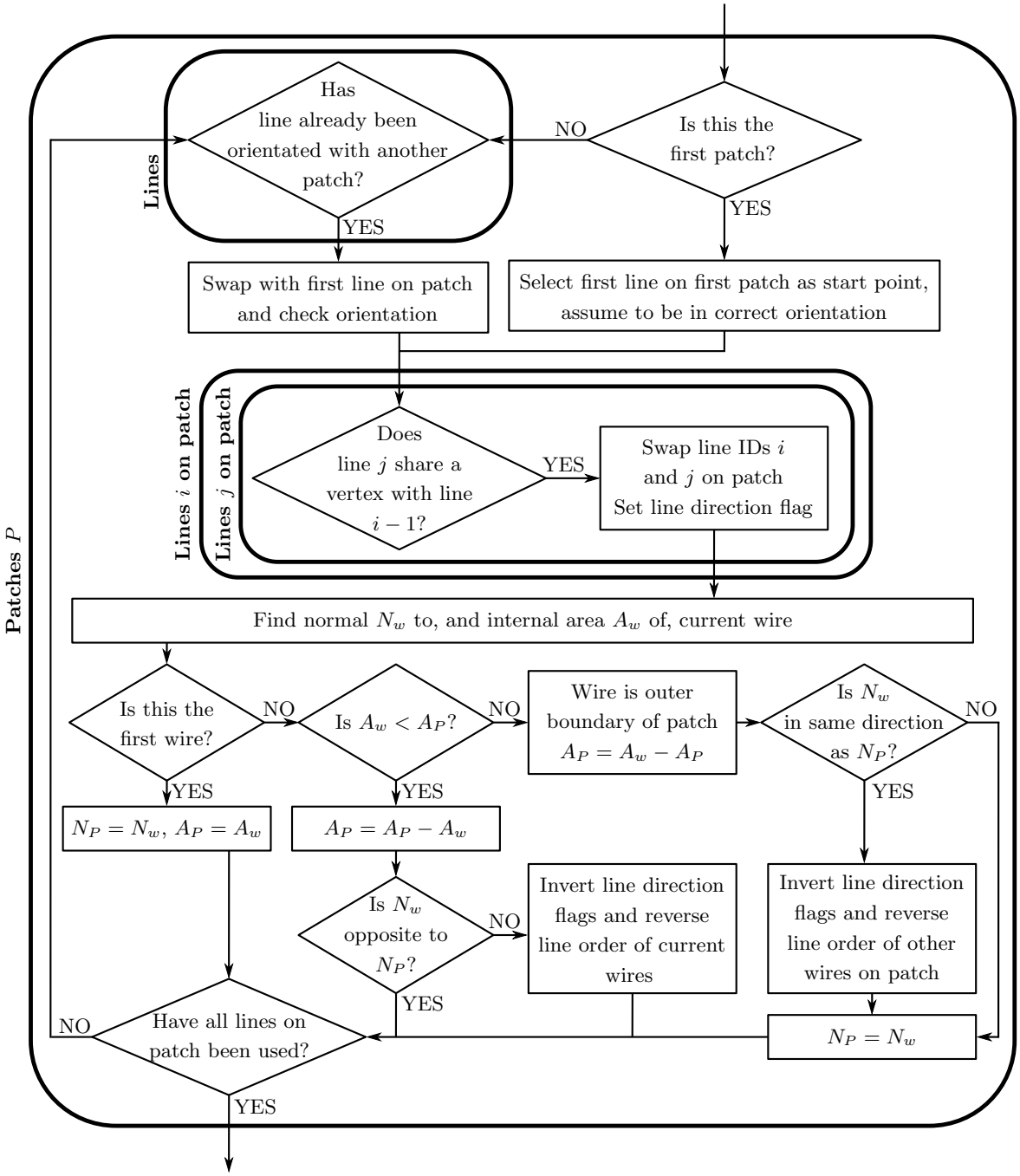


Figure 6.5: Flowchart showing order of the lines.

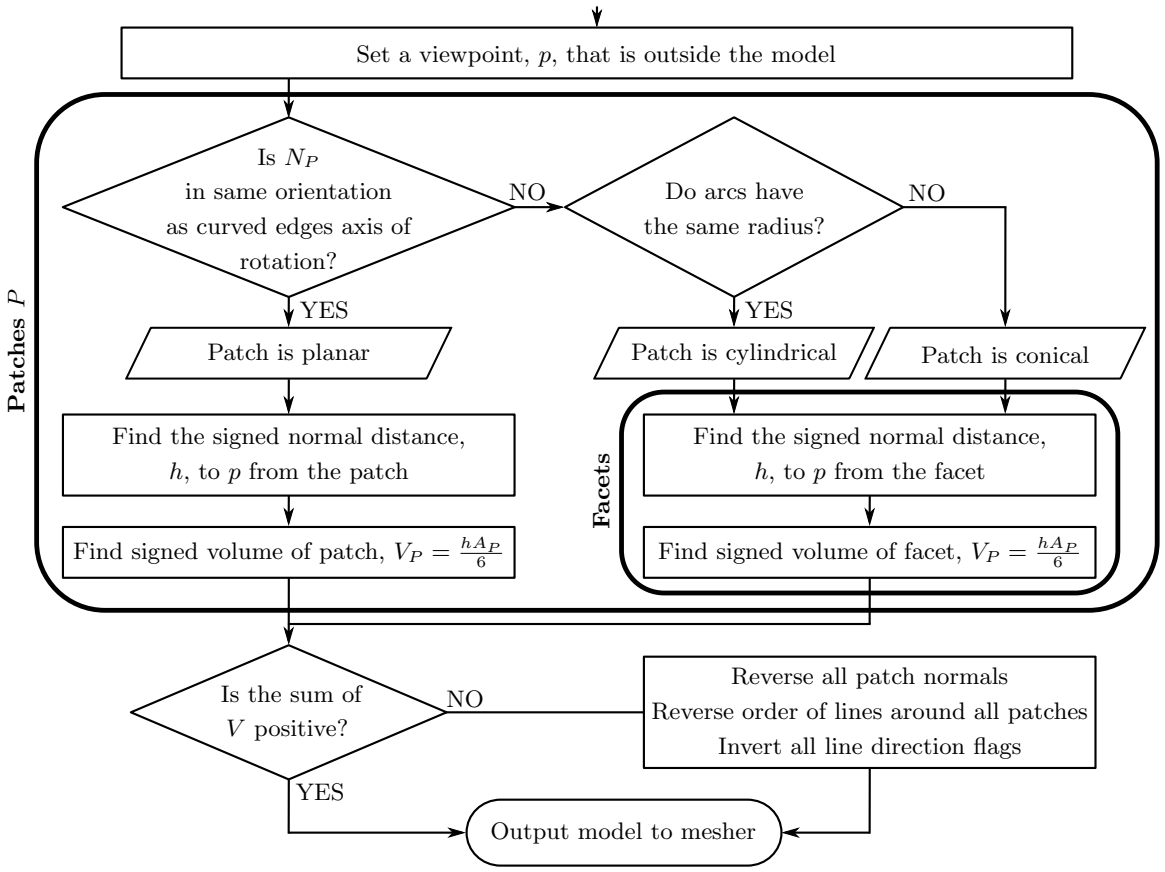


Figure 6.6: Flowchart showing orientation algorithm.

6.3 The mesh data structure

6.3.1 Introduction

Figure 6.7 shows the data structure that is created by the algorithm to store the elements that make up the model mesh. During mesh generation, each element is defined by mesh points that lie on the surface of the model and are therefore entirely dependent on the geometry. These are similar to nodes in that they are used to define elements but they always lie on the boundary of the element. Mesh points are used purely for convenience during the meshing process in case discontinuous elements need to be introduced. Once the mesh is finalised, nodes are generated from the mesh points forming an independent data structure. In this work the entire mesh is continuous and the location of the nodes will always be coincident with the mesh points. Therefore, for clarity, this thesis will refer only to nodes.

The information stored within each class is summarised in this section. All numbering systems start with zero.

A mesh can be fully defined using only the nodal coordinates and an array defining the nodal connectivity. However, if memory is available, extra details may be stored allowing increased access speed to all the data. Node based data structures typically use less memory but require more central processing unit (CPU) time for the additional traversals required. Modern computing technology means an ever-increasing amount of storage is becoming available and the current work can therefore concentrate primarily on speed; therefore a combination of node based and element based data structures can be used to make data access as fast as possible. However, care must be taken to ensure that any duplicated data sets are consistent.

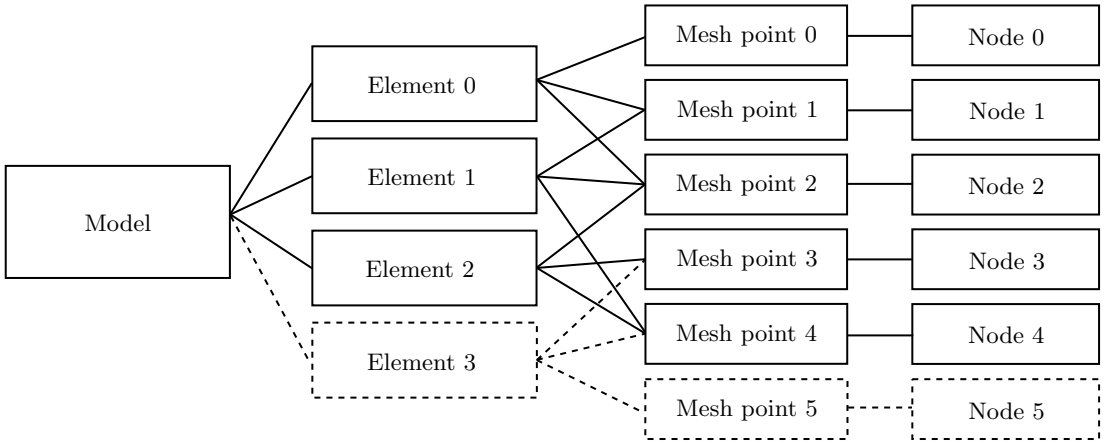


Figure 6.7: Mesh data structure.

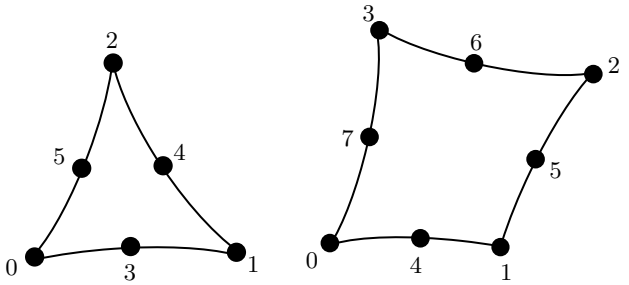
6.3.2 Elements

The mesh generator outputs a data set defining all the elements in the mesh shown in Table 6.4. This includes a reference to the patch on which the element exists, a flag defining the type of element and the ID numbers of the nodes around the element. The sides of each element are referred to as segments. The *element class* supports linear and quadratic, triangular and quadrilateral elements. The type of element is defined by the `elemType` field which stores the number of nodes on the element.

Table 6.4: Element class.

Field	**Description**
`int patchID`	ID for the patch on which the element lies.
`int elemType`	Type of element (8: quadrilateral; 6: triangular).
`int nodeID[8]`	ID numbers for the nodes around the element.
`float quality`	Element quality, Q.
`float size`	Nominal size of the element (average side length).
`int bcType[3]`	Type and (x, y, z) direction of boundary condition applied to element (0: traction; 1: displacement).
`float bcValue[3]`	Value of boundary condition, (x, y, z) components.

The node numbering conventions for triangular and quadrilateral elements, when viewed from outside the model, are shown in Figure 6.8. During mesh generation and smoothing only the corner nodes are considered; mid-side nodes are added later.



(a) Triangular element. (b) Quadrilateral element.

Figure 6.8: Node numbering.

Some triangular elements may be merged to form quadrilateral elements. This reduces the number of nodes and elements in the mesh, thereby accelerating the analysis procedure. However, merging elements may result in a reduction in mesh quality if not done carefully. Ideally, quadrilateral elements should be generated instead of a pair of triangular elements. However, due to the additional complexity involved in merging pairs of triangular elements into quadrilateral elements and the complications this will create in the re-meshing algorithm, quadrilateral elements will only be used on patches with a structured mesh.

6.3.3 Nodes

The *node class* stores the coordinates of each node and the number of elements that share the node along with their ID numbers, thus defining the element connectivity. This is summarised in Table 6.5. Nodes along lines are duplicated for different patches; these are linked in the `nodeID` field of the line class, given in Table 6.2, so that they can easily be merged during data output.

Table 6.5: Node class.

Field	Description
`float coords[3]`	(x, y, z) coordinates of node.
`int numElements`	Number of elements that share the node.
`int elementID[]`	ID numbers of the elements that share the node.
`int bcType[3]`	Type and (x, y, z) direction of boundary condition applied to node (0: traction; 1: displacement).

6.4 Global meshing factors

Several factors are defined globally for the mesh generation algorithm and are summarised in Table 6.6. The maximum adjacency and minimum grading ratios limit the size difference between adjacent elements around the corners or along lines respectively. These have been determined by numerical experimentation.

Trevelyan [30] suggests that, in two dimensions, an adjacent element should ideally never be more than twice the size of its neighbour. Therefore, as a general guideline, the grading ratios, $\lambda_{\max}$ and $\lambda_{\min}$, have been set to 2 and 0.5 respectively.

A nominal maximum segment size, $\hat{L}_{\max}$, is found from the size of the model bounding box and a user-defined mesh refinement factor, κ:

$$\hat{L}_{\max} = \frac{\kappa}{3}(\max(x) - \min(x)) + (\max(y) - \min(y)) + (\max(z) - \min(z)) \tag{6.1}$$

A segment connects two nodes to form the side of a pair of elements. The refinement factor, κ, may be set by the user to achieve an appropriate mesh density. The minimum number of elements around an arc, $C_{\min}$, is also limited to ensure sufficient refinement of the mesh around these important features.

Table 6.6: Global meshing factors.

Field	Symbol	Description
`float maxfactor`	$\lambda_{\max}$	Maximum adjacency ratio around vertices.
`float minfactor`	$\lambda_{\min}$	Minimum grading ratio along lines.
`float refine`	κ	Refinement factor.
`float maxelesize`	$\hat{L}_{\max}$	Nominal maximum element size.
`int circseg`	$C_{\min}$	Minimum number of elements around any arc.
`int rings`	n_R	Number of rings of points around arcs and return angles.

No user interaction should be required for meshing aside from defining the geometry, the boundary conditions and the mesh quality. For simplicity the general mesh density shall be presented to the engineer as options in a series of discrete steps ranging from coarse to fine. Table 6.7 shows the refinement used for coarse, medium and fine meshes.

Table 6.7: Mesh refinement factors.

	κ	$C_{\min}$	n_R
Coarse	0.5	16	2
Medium	0.2	16	3
Fine	0.1	24	4

6.5 Mesh quality measures

6.5.1 Element quality

To minimise error during discretisation of the mesh it is important to produce high quality elements. Many measures have been suggested for defining the quality, Q, of an element. The measures discussed here generate a value in the range $0 \leq Q \leq 1$ where $Q = 1$ indicates the highest quality element and 0 a fully collapsed element. The minimum element quality in the model can be regulated by setting a minimum permissible value of the chosen measure, $Q \geq Q_{\min}$.

Due to the meshing schemes used, all the quadrilateral elements encountered in this work will be rectangular. A simple quality measure can therefore be devised based on the aspect ratio of the element:

$$Q = \sqrt{\frac{L_{\min}}{L_{\max}}} \tag{6.2}$$

where $L_{\max}$ is the longest dimension of the element and $L_{\min}$ is the shortest dimension. The highest quality element is a square.

Triangular elements can be distorted in any way, therefore a more complex measure of Q is required for these elements. The properties used to construct the error measures are defined in Figure 6.9.

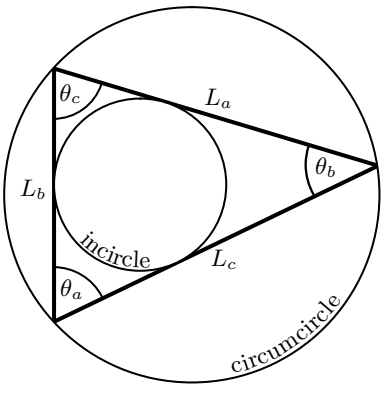


Figure 6.9: Element measures.

The radius of the inscribed circle within the element (incircle) and the radius of the circumscribed circle (circumcircle) are calculated using the following formulae where A is the area of the element.

$$R_i = \frac{2A}{L_a + L_b + L_c} \tag{6.3}$$

$$R_c = \frac{L_a L_b L_c}{4A} \tag{6.4}$$

The area of the element can be calculated using *Heron's area formula*:

$$A = \sqrt{s(s - L_a)(s - L_b)(s - L_c)} \tag{6.5}$$

where s is the semiperimeter of the element:

$$s = \frac{L_a + L_b + L_c}{2} \tag{6.6}$$

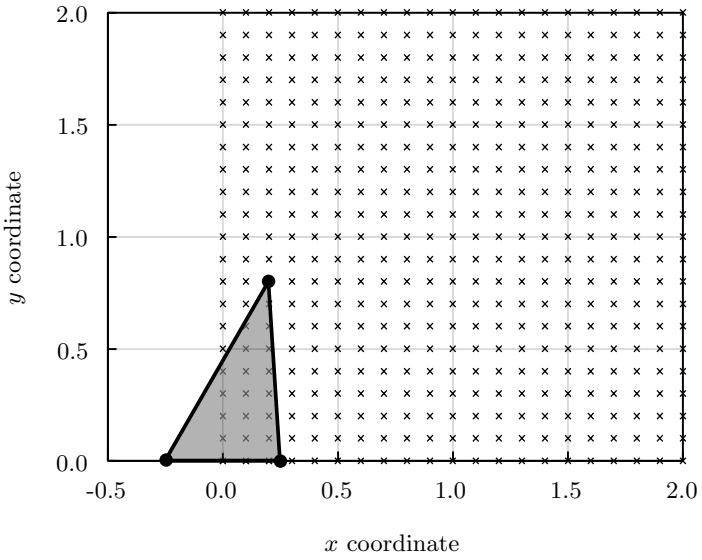


Figure 6.10: Quality measure testing scheme.

For comparison, each shape measure has been used to assess a systematically generated range of elements designed to give a good representation of all the shapes that may be encountered. Two nodes of each element were fixed at coordinates $(-0.25, 0)$ and $(0.25, 0)$ whilst the third node was successively defined at each point shown on Figure 6.10. The contour plots produced, for example Figure 6.11, show the element quality across this range and are symmetrical about both the x and y axes. For the highest quality element, an equilateral triangle, the third node should be located at $(0, \sqrt{3}/4)$.

Length Ratio

The length ratio is a measure of element quality, calculated from the ratio of the smallest side length of an element to the average side length [61].

$$Q_L = \frac{3 \min(L_a, L_b, L_c)}{L_a + L_b + L_c} \tag{6.7}$$

The length ratio is an acceptable measure of element quality for elements so long as the y coordinate of the third node exceeds 0.4 in the test case. This value can be scaled to fit any element. For elements that are close to fully collapsed (where all the edges lie on the same line), Q_L does not provide an appropriate quality measure. The accuracy is most misleading when the third node is positioned closest to (0,0), where the measure tends to 0.75 instead of 0. This problem can clearly be seen on Figure 6.11, which shows the range of shape ratios produced by analysing a set of elements produced using the grid shown in Figure 6.10. The length ratio is best used in conjunction with another shape measure to produce a

more suitable quality measure.

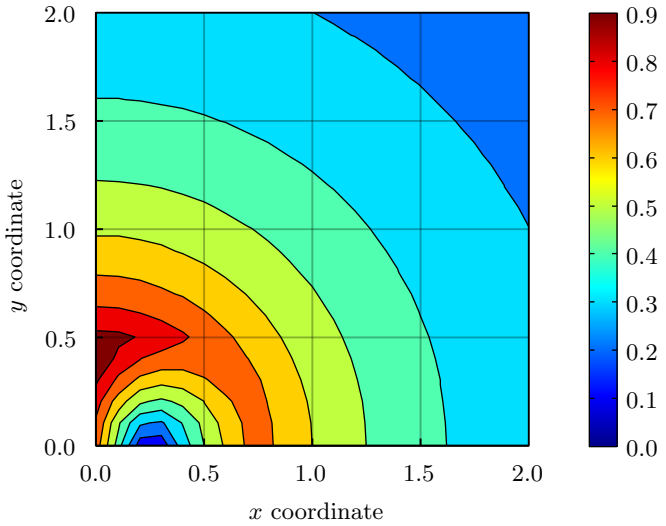


Figure 6.11: Length ratio, Q_L, for a range of elements.

Angle ratio

The angle ratio, suggested by the author as an improvement on the length ratio, is the ratio of the smallest angle in the element to the average angle ($\pi/3$ radians).

$$Q_\theta = \frac{3\min(\theta_a, \theta_b, \theta_c)}{\pi} \tag{6.8}$$

The angle ratio gives a more reliable estimate of the element quality than the length ratio. These measures complement each other well and could be used together to more accurately assess element quality. The angle ratio successfully identifies near fully collapsed elements as poor quality and provides a tight, if slightly irregular, set of contours about the optimal element shape, as can be seen in Figure 6.12.

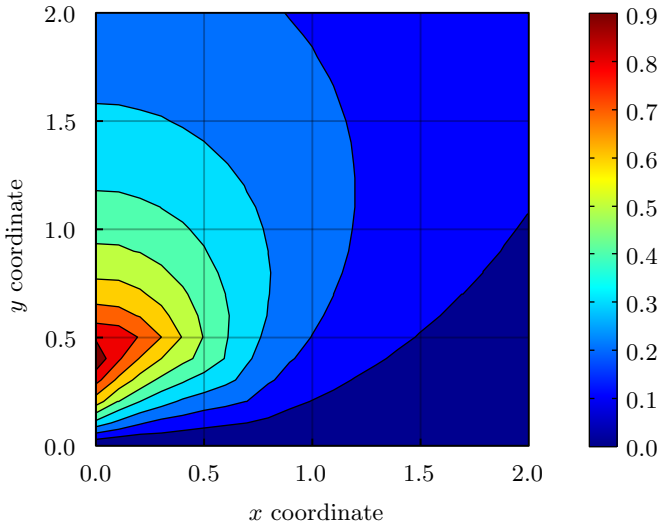


Figure 6.12: Angle ratio, Q_θ, for a range of elements.

Normalised aspect ratio

The element aspect ratio is the ratio of the radius of the incircle to the length of the longest side [61]. This ratio has been multiplied by $6/\sqrt{3}$ to scale it to a value between 0 and 1.

$$Q_A = \frac{12A}{\sqrt{3}(L_a + L_b + L_c)\max(L_a, L_b, L_c)} \tag{6.9}$$

The normalised aspect ratio produces circular contours about the optimum element shape, as shown in Figure 6.13. It provides an appropriate measure of the element quality for any amount of distortion.

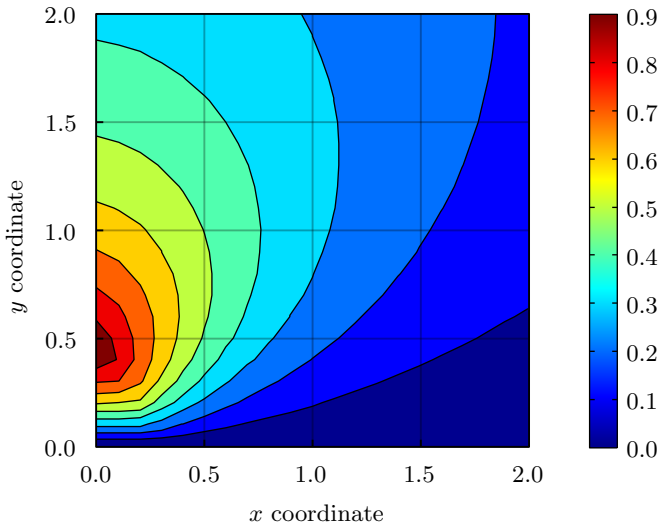


Figure 6.13: Normalised aspect ratio, Q_A, for a range of elements.

Normalised circle ratio

The circle ratio is a two-dimensional adaptation of the radius ratio discussed by Topping *et al.* [61] for assessing tetrahedral elements used in three-dimensional finite element (FE) analysis. It is the ratio of the radius of the incircle to the radius of the circumcircle. This ratio has been multiplied by 2 to scale it to a value between 0 and 1.

$$Q_C = \frac{2R_i}{R_c} = \frac{16A^2}{L_a L_b L_c (L_a + L_b + L_c)} \tag{6.10}$$

The circle ratio generates a very similar contour plot to the aspect ratio. The aspect ratio gives a marginally more useful measure for low quality elements but for an acceptable element quality they are near identical. The circle ratio requires marginally less computation than the aspect ratio and therefore is preferable where fast computation is required. To provide the same mesh quality, the minimum circle ratio must be slightly higher than for the aspect ratio. However, testing will be required to establish exactly how much the element quality affects the analysis results and the minimum quality measure will need to be fine-tuned.

Summary

The normalised circle ratio, Q_C, has been selected as the most appropriate element quality measure for this work. To simplify the notation, the element quality is often referred to as Q throughout this thesis. However, wherever element quality is discussed, Q_C has been applied to work out any values.

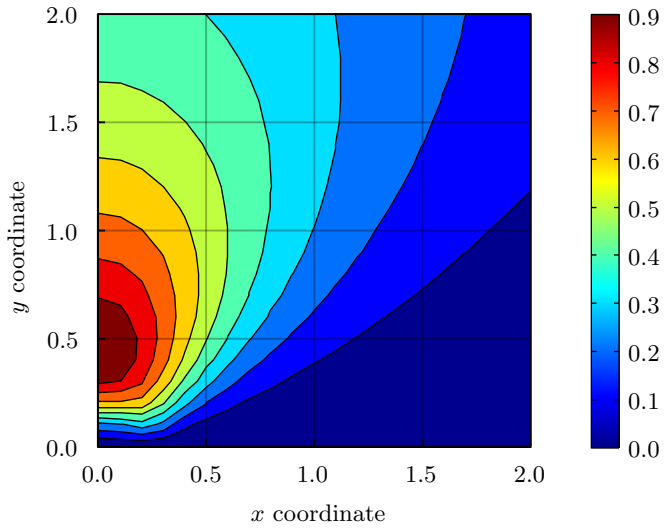


Figure 6.14: Normalised circle ratio, Q_C, for a range of elements.

6.5.2 Mesh quality

The mean, $\bar{Q}$, and standard deviation, $\tilde{S}$, of the accumulated single element measures is highly representative of the global mesh quality:

$$\bar{Q} = \frac{1}{n_E} \sum_{k=1}^{n_E} Q_k \tag{6.11}$$

$$\tilde{S} = \frac{1}{n_E} \sum_{k=1}^{n_E} \left(Q_k - \bar{Q}\right)^2 \tag{6.12}$$

where n_E is the number of elements in the assessed area.

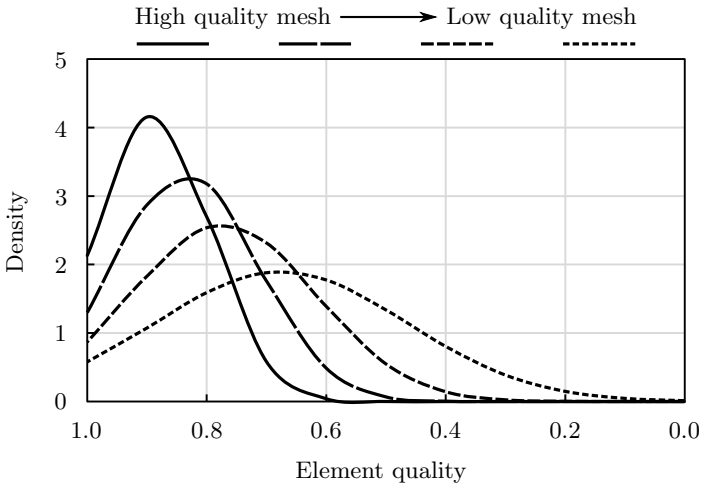


Figure 6.15: Distribution of element quality.

Figure 6.15 shows four normal distributions of element quality for a simple mesh of 150 elements assessed using Q_C. It should be noted that the distributions shown on Figure 6.15 are the true normal distribution for the meshes, calculated from $\bar{Q}$ and $\tilde{S}$. To preserve mesh quality we define measures $\bar{Q}_{\min}$ and $\tilde{S}_{\max}$ to be the minimum acceptable mean element quality and the maximum acceptable standard deviation of element quality respectively. If the mesh quality falls outside of these criteria, further

smoothing must be carried out.

6.6 Meshing strategy

6.6.1 Mesh refinement

The meshes produced by the algorithms presented in this thesis are refined according to the initial geometry of the model. Geometric recognition is used with the aim of predicting where high stress gradients are likely to occur so that the mesh can be refined in these areas. This contrasts with many schemes which use an initial coarse analysis to inform refinement of the mesh. As the geometries encountered in this work are generally simple and larger errors are permitted during re-analysis, this is not necessary and time can be saved by moving immediately to a suitable mesh.

6.6.2 Initial segment sizes

The mesh generator initially defines the maximum segment size, δ_{ij}, for each line, i, at each vertex, j, as shown in Figure 6.16.

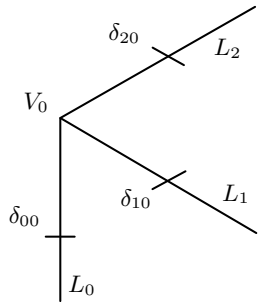


Figure 6.16: Mesh grading factors for lines around a vertex.

The factors δ_{ij} are calculated by first finding the internal angle, β, between the two patches which meet at line i. This is done by comparing the patch normals. A scaling factor, γ, is defined depending on the value of β:

$$\gamma = \begin{cases} 1.0 & \text{if}(\beta < \pi/2) \\ 0.5 & \text{if}(\pi/2 \leq \beta < \pi) \\ 0.2 & \text{if}(\beta \geq \pi) \end{cases} \tag{6.13}$$

A nominal segment size, $\hat{\delta}_{ij}$ can be found for vertices j at each end of line i:

$$\hat{\delta}_{ij} = \gamma \hat{L}_{\max} \tag{6.14}$$

where $\hat{L}_{\max}$ is the nominal maximum element size calculated using equation (6.1).

The grading factors for the appropriate end of all the lines adjoining the current line are defined in this operation. However, the values on the current line are not defined as they depend on the other lines around the vertex. If a new grading factor is calculated where one has already been defined, only the lowest value is stored. This satisfies the refinement criteria by generating suitably small elements in the region.

To avoid rapid grading between elements of different sizes around a vertex, j, it is ensured that:

$$\delta_{ij} \leq \lambda_{\max} \delta_{\min,j} \tag{6.15}$$

where $\delta_{\min,j}$ denotes the smallest segment around vertex j. If the values of δ_{ij} at both ends of line i are small in comparison to its length, the line is split in half during segment distribution so that they can grade out in the centre. Additional grading factors are defined at the centre of the line.

6.6.3 Segment grading

Once the values of δ_{ij} have been computed for every line in the model, the grading ratio, λ, between adjacent segments along each line is calculated. A geometric progression is assumed for grading the segments along line i where each term corresponds to the length of a segment. For clarity the i subscripts have been omitted, $\delta_{\max}$ therefore refers to the largest of the two values of δ on line i and $\delta_{\min}$ the smallest.

$$l = \delta_{\max}(1 + \lambda + \lambda^2 + \lambda^3 + ... + \lambda^{n-1}) \tag{6.16}$$

where l is the line length.

By multiplying both sides by $(1 - \lambda_i)$ equation 6.16 can be rearranged to give:

$$l = \delta_{\max}\left(\frac{1 - \lambda^n}{1 - \lambda}\right) \tag{6.17}$$

It should be noted that the nth term in the series is known:

$$\delta_{\min} = \delta_{\max}\lambda^{n-1} \tag{6.18}$$

The grading factor, λ, can now be found by substituting equation (6.18) into equation (6.17) and rearranging for λ:

$$\lambda = \frac{l - \delta_{\max}}{l - \delta_{\min}} \tag{6.19}$$

If $\lambda < \lambda_{\min}$, then it is increased to equal $\lambda_{\min}$. The number of segments, n, along the line can be calculated by rearranging equation (6.18). To force n to be an integer it is rounded up.

$$n = \text{ceil}\left(\frac{\ln(\delta_{\min}/\delta_{\max})}{\ln(\lambda)} + 1\right) \tag{6.20}$$

Under certain conditions n may need to be modified. If $n < 2$, $n = 2$ and if i is an arc and $n/\theta < C_{\min}/(2\pi)$, $n = \text{ceil}(\theta C_{\min}/2\pi)$. Consistency is also checked across patches that are to use a structured quadrilateral mesh.

The apparent length of i, $\hat{l}$, can now be calculated by using equation (6.17). This is used in conjunction with the actual length to scale $\delta_{\max}$ so that a whole number of segments fit along the line.

$$\delta'_{\max} = \delta_{\max} l/\hat{l} \tag{6.21}$$

If $\delta_{\max}$ has been updated then $\delta_{\min}$ must also be adjusted using equation (6.18). If the grading scheme has generated an excessive number of elements along a line, the line is split in half so that the size of element can grade out to $\delta_{\max}$ in the centre.

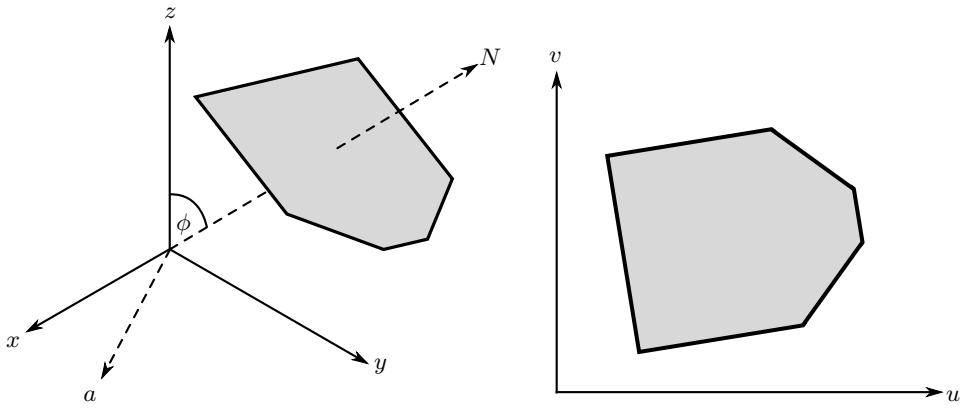
Whenever δ_{ij} is updated, all the values of δ_{ij} around vertex j must be rechecked against $\lambda_{\max}$. The grading along associated lines must the be checked again. This process continues to iterate until no changes occur.

6.6.4 Transformation into two dimensions

To simplify the generation of an unstructured mesh across each patch, a bidirectional mapping is applied to transform each patch from the Cartesian (x, y, z) coordinate system onto a planar parametric space (u, v). These mappings have been adapted by the author from those suggested by Lo [105].

Plane surface

For a plane patch the transformation is trivial. The patch is simply rotated through the angle ϕ, so that the outward normal aligns with the z-axis. This is shown in Figure 6.17.



(a) Cartesian surface. (b) Parametric surface.

Figure 6.17: Transformation of a plane patch.

The rotation matrix is given as:

$$[R] = \begin{bmatrix} a_x^2 + (1 - a_x^2)\cos\phi & a_x a_y(1 - \cos\phi) & a_y \sin\phi \\ a_x a_y(1 - \cos\phi) & a_y^2 + (1 - a_y^2)\cos\phi & -a_x \sin\phi \\ -a_y \sin\phi & a_x \sin\phi & \cos\phi \end{bmatrix} \tag{6.22}$$

where ϕ is the angle between the outward normal of the patch, N, and the z axis and a is the unit axis of rotation calculated from the cross product of the outward normal and the z axis. The patch may then be transformed by pre-multiplying the location of any point on the patch with the rotation matrix:

$$\begin{Bmatrix} u \\ v \\ w \end{Bmatrix} = \begin{bmatrix} \\ R \\ \\ \end{bmatrix} \begin{Bmatrix} x \\ y \\ z \end{Bmatrix} \tag{6.23}$$

The w coordinate will be identical for all points that lie on the patch and is therefore only calculated once. It is stored by the algorithm and used when mapping back into Cartesian coordinates:

$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \begin{bmatrix} \\ R \\ \\ \end{bmatrix}^{-1} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} \tag{6.24}$$

Cylindrical surface

A cylindrical surface is transformed using a local axis system (a, b, c) where a, b and c are unit vectors in the directions shown in Figure 6.18 centred on point O. P defines a point anywhere on the surface and M defines the projection of point P on the axis of rotation of the surface, a.

In parametric space u is the distance around the circumference of the cylinder and v is the distance along the axis of rotation such that:

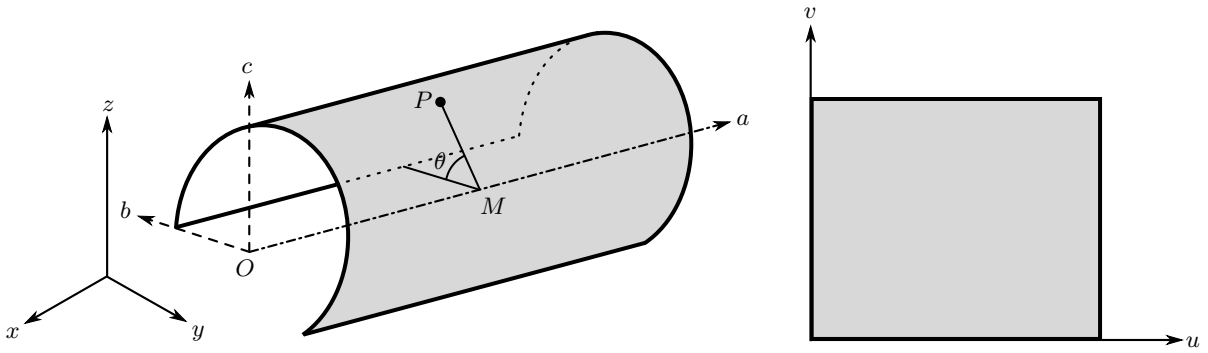
$$u = r\theta \tag{6.25}$$

$$v = OP \cdot a \tag{6.26}$$

where r is the radius of the cylinder and θ is given by:

$$\theta = \cos^{-1}\left(\frac{MP \cdot b}{r^2}\right) \tag{6.27}$$

$$MP = OP - (OP \cdot a)a \tag{6.28}$$



(a) Cartesian surface. (b) Parametric surface.

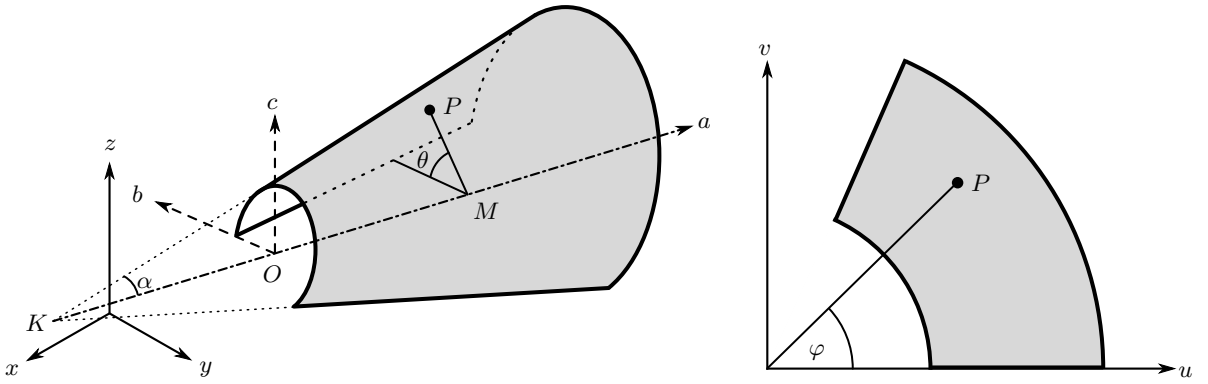
Figure 6.18: Transformation of a cylindrical patch.

The inverse of the transform for a point on the (u, v) plane is given by:

$$P = O + av + b\cos\left(\frac{u}{r}\right) + c\sin\left(\frac{u}{r}\right) \tag{6.29}$$

Conical surface

A cone is transformed using a similar method to that for the cylinder. The variables for a cone are assigned as shown in Figure 6.19. P denotes a point anywhere on the surface.



(a) Cartesian surface. (b) Parametric surface.

Figure 6.19: Transformation of a conical surface.

$$\theta = \cos^{-1}\left(\frac{MP \cdot b}{MP}\right) \tag{6.30}$$

$$MP = OP - (OP \cdot a)a \tag{6.31}$$

In parametric space, the angle φ is given by:

$$\varphi = \frac{\|MP\|\theta}{\|KP\|} \tag{6.32}$$

Hence, the parametric coordinates of point P are given by:

$$u = \|KP\| \cos\varphi \tag{6.33}$$

$$v = \|KP\| \sin\varphi \tag{6.34}$$

The inverse of the transform can be calculated using the following set of equations:

$$r = \sqrt{u^2 + v^2} \tag{6.35}$$

$$\varphi = a\cos\left(\frac{u}{r}\right) \tag{6.36}$$

$$\theta = \frac{\varphi}{\sin\alpha} \tag{6.37}$$

$$P = O + ar\cos\alpha + br\sin\alpha\cos\theta + cr\sin\alpha\sin\theta \tag{6.38}$$

Data structure

The data structure for the transformed two-dimensional patches is the same as for their three-dimensional equivalents with the addition of the variables used in the transformations shown in Table 6.8.

Table 6.8: Transformation data.

Field	Symbol	Description
Plane surface		
`float w`	w	w coordinate of rotated patch
`float R[3][3]`	$[R]$	Rotation matrix
`float Ri[3][3]`	$[R]^{-1}$	Inverse of rotation matrix
Cylinder and Cone		
`float a[3]`	a_x, a_y, a_z	Unit axis (axis of plane)
`float b[3]`	b_x, b_y, b_z	Unit axis (direction of start of surface)
`float c[3]`	c_x, c_y, c_z	Unit axis ($a \times b$)
`float M[3]`	O_x, O_y, O_z	Coordinates of origin of axis system
`float K[3]`	K_x, K_y, K_z	Coordinates of tip of cone
`double alpha`	α	Half angle in tip of cone

6.6.5 Mesh generation across plane surfaces

Node distribution

Nodes are generated along the lines around each transformed patch using the final values of δ_1 and δ_2 and the geometric progression defined in equation 6.16. If the line is subdivided to allow elements to grade out in the centre, each half is treated separately.

Nodes are systematically generated across the interior of the patch. These are concentrated around

key features to ensure sufficient refinement of the mesh. A double ring of nodes is generated around a re-entry corner, as illustrated in Figure 6.20(a); the number of nodes in the ring is calculated from the severity of the angle. A single node is placed so that it bisects any corners with an internal angle greater than $\pi/3$ radians and less than π radians, as shown in Figure 6.20(b). The distance from the feature to these nodes is determined from the average of the end segment lengths of the line that meet at the feature. A ring of nodes is located around the inside of a concave arc, as illustrated in Figure 6.20(c). The number of nodes placed around an arc is equal to the number of segments along the arc and the distance is dictated by the size of the segments. A single, sparser line of nodes is also generated along each straight line and convex arc. Additional nodes may be generated in random positions across the interior of each patch if a large, node free area is detected. The number of randomly placed nodes is based on the number of segments around the patch boundary and the number of internal nodes already created.

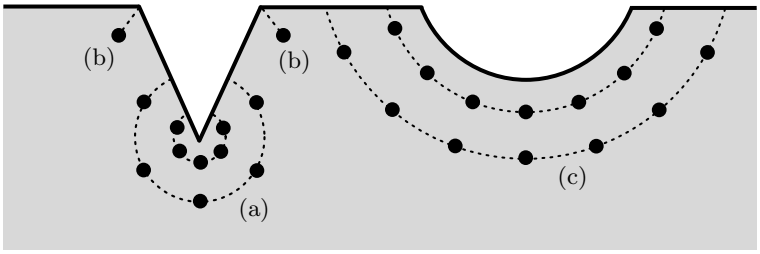


Figure 6.20: Distribution of nodes around geometrical features.

Initial triangulation

A constrained Delaunay triangulation is used to generate a mesh across the patch. This connects all the nodes on a patch to form a triangular mesh. An initial unstructured mesh is of relatively low quality. Further processing is required to remove distorted elements and improve the quality of the mesh. This is called smoothing and is applied in several steps, as detailed in Section 6.6.6. Structured meshes initially produce higher quality elements than unstructured meshes, due to this, and to preserve the structure, no smoothing is carried out after the initial triangulation across structured meshes.

6.6.6 Mesh smoothing

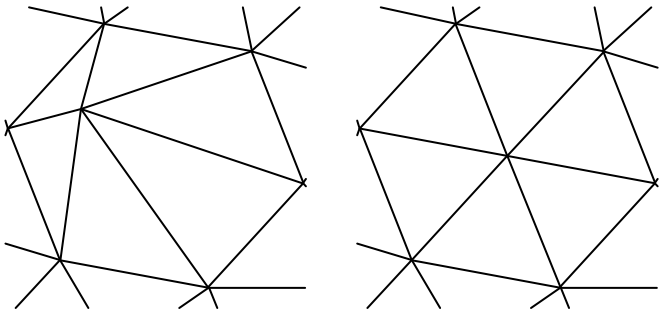
Several algorithms are applied to improve the quality of a triangular mesh. The algorithms detailed later in this section can be applied to the mesh in any order and may be repeated to further improve the mesh. To minimise the amount of computation required in producing a high quality mesh for general problems, the author recommends the following:

1. Laplacian smoothing (2 iterations).
2. Element collapse.
3. Triplet removal.
4. Further Laplacian smoothing (2 iterations).
5. Element sub-division.

Laplacian smoothing

The Laplacian smoothing algorithm iterates around all the nodes within the boundaries of each patch. The u and v coordinates of the nodes connected via a single element to the current node are found and averaged to give a new location for the current node. Several iterations of this process can be carried out

to remove a significant amount of the element distortion from the initial mesh. The process is illustrated in Figure 6.21.

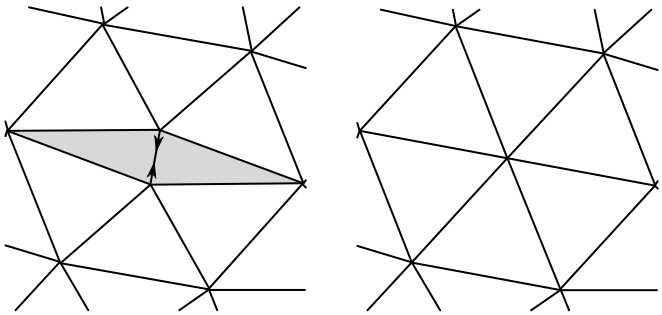


(a) Original mesh. (b) Smoothed mesh.

Figure 6.21: Laplacian smoothing.

Element collapse

Poor quality elements are detected using the normalised circle ratio quality measure discussed in Section 6.5. These elements are then removed by collapsing their shortest side and merging the nodes at each end to the centre of the removed segment. If either end of this segment lies on a line then the nodes are merged at this end of the segment. If the shortest side of an element is on the patch boundary, it is not removed, as this change would need to propagate onto the adjacent patch. This process also removes the adjoining element and may result in the removal of some small elements that are not distorted; however, this has very little impact on the overall mesh quality and reduces the number of elements, thus speeding up analysis. The process of element removal is illustrated in Figure 6.22.

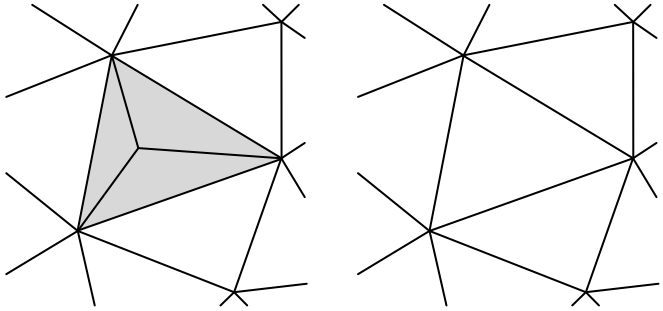


(a) Original mesh. (b) Smoothed mesh.

Figure 6.22: Element collapse.

Triplet Removal

An example of an element triplet is shown in Figure 6.23(a). Element triplets produce poor quality elements and result in a high element density in very localised areas. The mesh density of the surrounding elements is sufficient, hence the extra refinement created by the triplet is deemed unnecessary. Any node surrounded by three elements is removed along with the surrounding elements, which are replaced by a single large element.

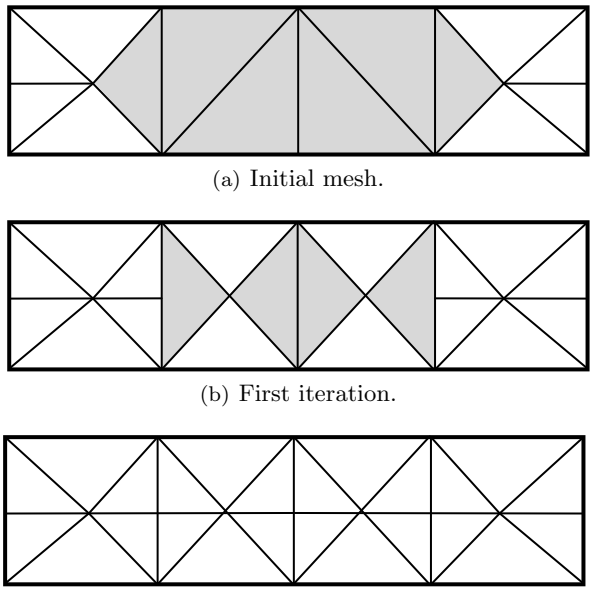


(a) Original mesh. (b) Smoothed mesh.

Figure 6.23: Element triplet removal.

Element splitting

Any elements that span across the entire patch are sub-divided. This reduces the risk of a large stress gradient across a single element, such as when an element spans the height of the side of a cantilever. Two iterations of this algorithm are required to ensure all spanning elements are caught, as shown in Figure 6.24.



(a) Initial mesh.

(b) First iteration.

(c) Second iteration.

Figure 6.24: Element sub-division.

6.6.7 Mesh generation across conical and cylindrical surfaces

Conical and cylindrical surfaces generally have smooth stress contours aligned with the patch. A structured quadrilateral mesh therefore provides visually better contours across these surfaces. The structured mesh is generated by constructing a grid across these patches which joining nodes on opposing sides of the patch. The segment grading algorithms ensure that there are an equal number of appropriately spaced nodes on opposing sides of the patch. This grid is used to form elements with nodes at the intersections of the grid. The mid-side nodes are then generated.

6.6.8 Transformation into three dimensions

Once the two-dimensional mesh has been finalised, nodes are generated at the centre of each segment. If the segment lies on a curved line, the node is positioned on the line. The three-dimensional location of each node is then calculated using a transformation from the parametric space (u, v) to the Cartesian model space (x, y, z). The global data structure is updated to include the newly transformed nodes and the elements that connect them. As each line appears on more than one patch, duplicate nodes exist along them. The ID numbers of these nodes are stored in the `nodeID` field of the line class. This allows the algorithm to quickly establish which nodes are on top of each other and on which line they lay. The ID numbers of the elements connected to each node are also stored. These are required during re-meshing.

6.6.9 Data output

The nodes and element topology are output to the solver. During this process, duplicated nodes around the boundaries of each patch are merged with those on adjacent patches.

6.6.10 Summary

A new mesh-generation algorithm has been developed for the current work. The algorithm ensures that sufficient control can be maintained over the mesh and data structure during the meshing and re-meshing procedures. The overheads are also reduced by generating only the data required for analysis and subsequent re-analysis. All the elements generated during meshing are continuous quadratic serendipity elements and may be either triangular or quadrilateral. Conical and cylindrical surfaces are meshed with a structured mesh whilst a constrained Delaunay triangulation scheme [85] is used to generate triangular meshes across plane surfaces. An example of a mesh generated is shown in Figure 6.26. Finer meshes can be generated as required according to user preferences.

The shape of the elements used in the mesh directly affects the accuracy of an integration scheme of prescribed order. The initial mesh is therefore generated from high quality quadratic boundary elements (BEs) and refined in areas where high stress concentrations have been predicted. The predictions are based on heuristics applied to the geometry of the initial model.

The normalised circle ratio Q_C, described in Section 6.5, has been adopted for determining the quality of triangular elements. Quadrilateral elements are only generated on cylindrical or conical surfaces and, using the procedures followed by the meshing and re-meshing algorithms, will always be of sufficient quality. If a measure were to be required the element aspect ratio could be used, along with some constraint on the internal angles.

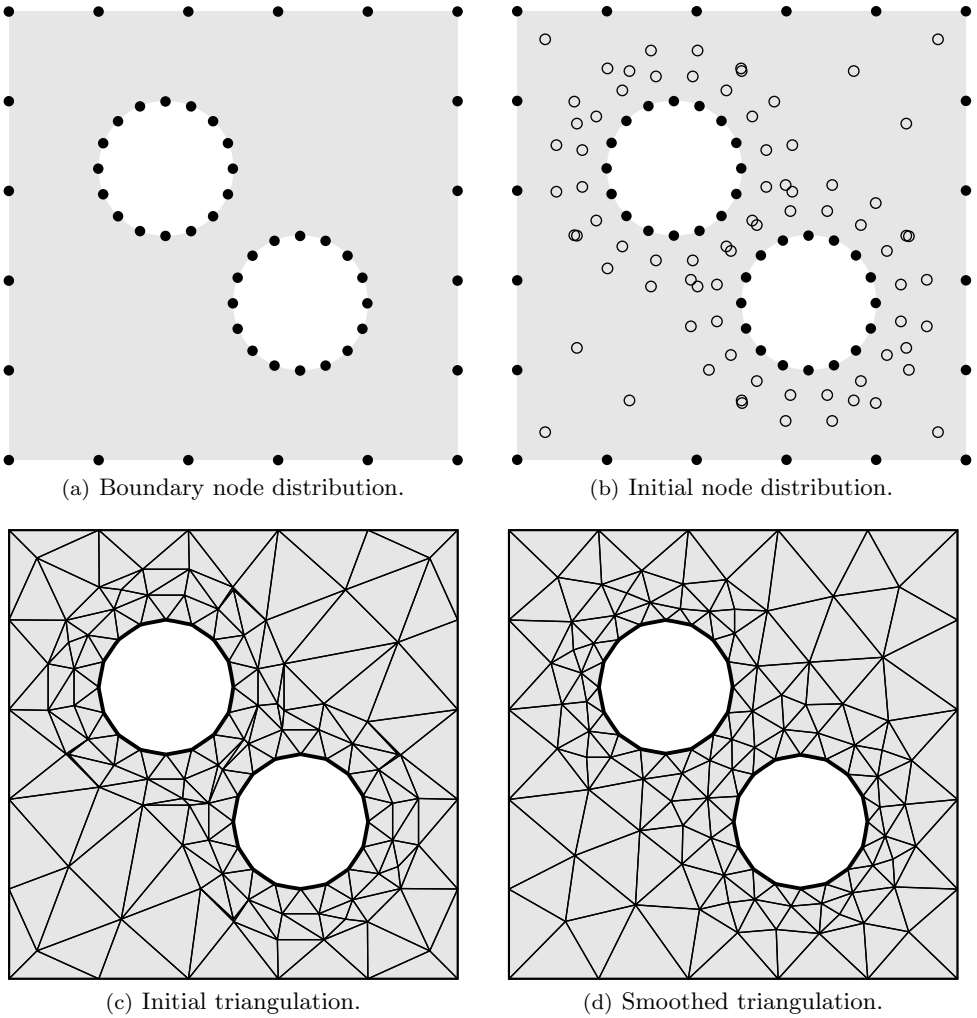
The main steps involved in the initial unstructured mesh generation strategy over a single patch are summarised below. A structured mesh would simply omit step 6.

1. Define segment sizes, δ_{ij}, at both ends of every line and use to generate a geometric progression.
2. Convert each patch into the two-dimensional domain.
3. Distribute boundary nodes around each patch based on δ_1, δ_2, λ and n for each line, as shown in Figure 6.25(a).
4. Distribute internal nodes based on geometric features as illustrated in Figure 6.25(b).
5. Carry out a constrained Delaunay triangulation of the nodes, as shown in Figure 6.25(c).
6. Smooth the mesh to obtain Figure 6.25(d).
7. Generate mid-side nodes.
8. Transform the mesh back into the three-dimensional domain.

The patches are then combined to form the three-dimensional mesh such as that shown in Figure 6.26.

Nodes are generated and the final model is sent to the solver for analysis. The entire process is covered in more detail by the flowchart given in Figure 6.27.

If the user changes the model, re-meshing must be carried out before the model can be re-analysed. The re-meshing procedure is detailed in Chapter 7.3.



(a) Boundary node distribution.

(b) Initial node distribution.

(c) Initial triangulation.

(d) Smoothed triangulation.

Figure 6.25: Meshing example.

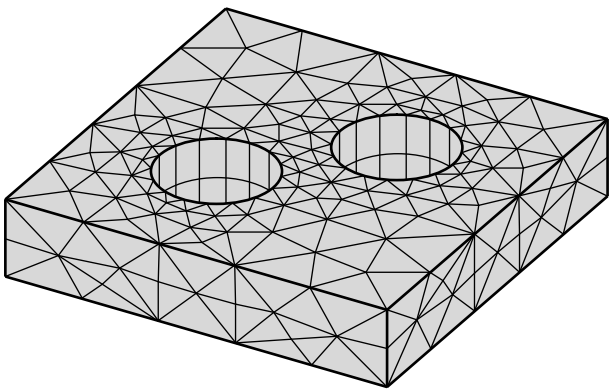


Figure 6.26: Three-dimensional mesh example.

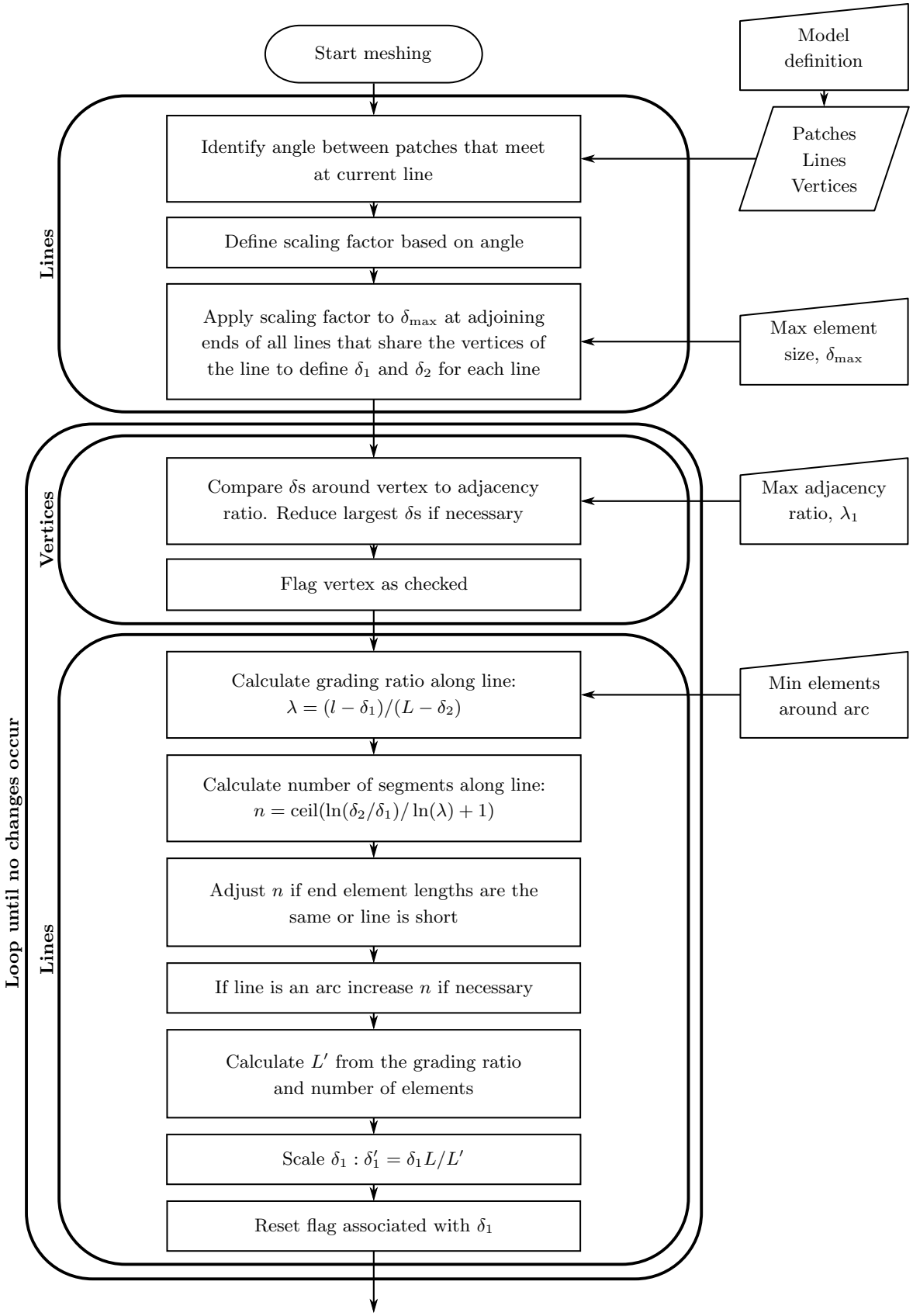


Figure 6.27: Meshing flowchart part 1.

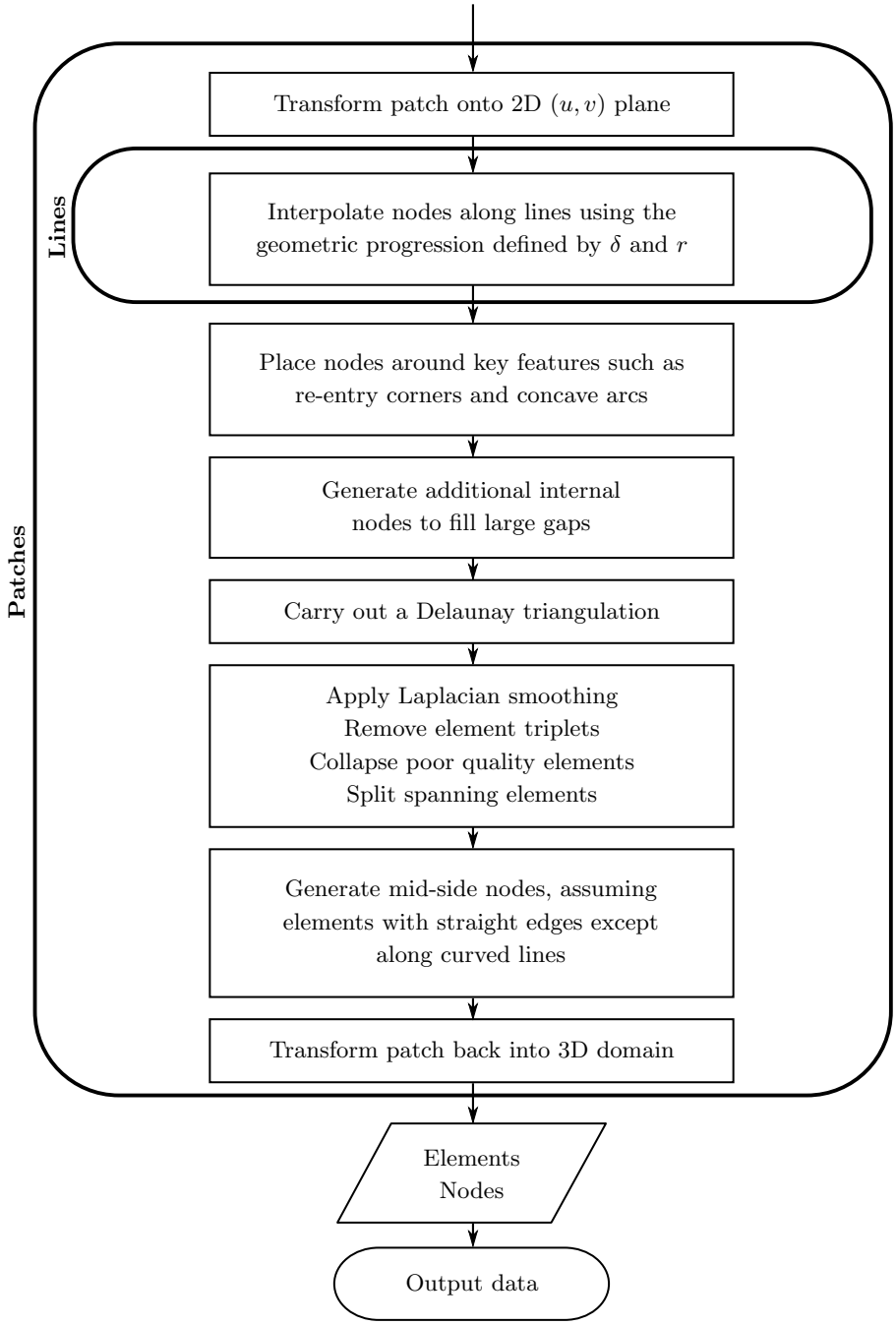


Figure 6.28: Meshing flowchart part 2.

6.7 Parallelising the mesh generation

An early parallelisation algorithm for mesh generation is presented by Lohner *et al.* [167] who split the domain into several sub-domains. Each sub-domain is then sent to a different processor to be meshed. This technique could easily be applied to the algorithm proposed in this thesis by sending each patch to a different CPU. However, load balancing techniques would need to be employed to intelligently spread patches of variable size around the CPUs. The meshing of small models, such as those encountered in this work, takes a trivial amount of time. The current meshing scheme has therefore not been parallelised as the additional overheads would be detrimental to the run time.

Chapter 7

Re-meshing to accommodate geometrical changes

"To raise new questions, new possibilities, to regard old problems from a new angle, requires creative imagination and marks real advance in science."

Albert Einstein

7.1 Introduction

A re-meshing scheme is required to update the mesh as the user adjusts parts of the model. This scheme should maintain an acceptable mesh quality whilst minimising the number of elements and nodes that are changed with each update. This limits the number of values that need to be re-calculated in the system matrix and hence accelerates re-analysis.

The updating scheme assumes all elements are continuous. It is also assumed that a series of small changes are being carried out on the model by the user, for example, moving a hole in a plate. The mesh will need to be updated successively after every small movement of the hole. If a larger change is made to the model, the global mesh quality becomes too degraded and the entire mesh will need to be regenerated.

7.2 Additions to the data structure

A geometric update may affect the geometry in a number of predefined ways which are interpreted differently by the re-meshing algorithm. Four situations can be identified for updating each patch, as shown in Figure 7.1:

(a) No change is made to the patch: No change is made to the mesh.
(b) The patch is translated: All the nodes and elements on the patch are translated through the same vector. The integrals for the cases where point p and element e both lie on the patch therefore do not need to be recalculated but other integrals will need to be updated.
(c) The patch is distorted out of plane: All elements on the patch are updated. The mesh is regenerated across the patch and all integrals will need to be recalculated.
(d) The patch is distorted but remains in plane: Only some elements on the patch require updating, the rest of the mesh remains unchanged.

As during initial mesh generation, each patch is treated individually during re-meshing.

A similar scheme is applied to updated elements. If a group of elements is translated through the same vector, the integrals between this group of elements remain the same and do not need to be updated;

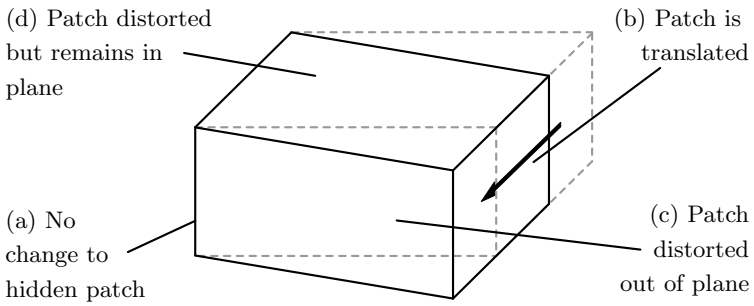


Figure 7.1: Geometric update of a block showing changed patches.

they must however be recalculated for the rest of the mesh. Elements distorted both in plane or out of plane have moved independently and therefore require the associated integrals to be recalculated across the entire mesh.

A line may be updated in any one of five ways depending on the type of geometric change, as shown in Figure 7.2:

(a) No change is made to the line: No change is made to the node distribution.
(b) The line is translated: The nodal distribution is translated.
(c) The line is an arc and has changed length: The nodal distribution is fully recalculated.
(d) The length of the line has changed but is on the same axis: Only some of the distribution is updated.
(e) The axis of the line has changed: The original distribution is scaled.

If a rotational translation is applied to a line, the nodal distribution along it may be updated using (b). However, rotational translation is not currently supported and the distribution is instead updated using (c) or (e) for arcs and straight lines respectively.

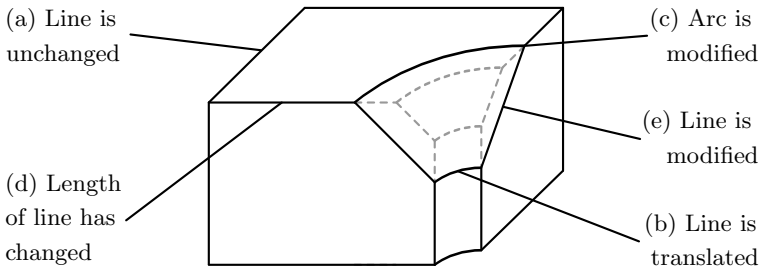


Figure 7.2: Geometric update of a model showing changed lines.

Several entries need to be added to the data structure to accommodate the additional flags and data required for re-meshing. These are summarised in Table 7.1.

The update flags F_{Lu} and F_{Pu} that define whether the geometric entity has been updated by the user since the mesh was last regenerated are summarised in Table 7.2. Updates to the model are generated in the modelling algorithm and inform the re-meshing algorithm. When the re-meshing algorithm is called, the flags are checked to establish which areas of the mesh need updating. During re-meshing this algorithm flags updated nodes F_{Nu} and elements F_{Eu}; these flags inform the re-analysis code.

Table 7.1: Additional re-meshing data.

Feature	Field	Symbol	Description
Vertex	`float translation[3]`	T_V	Vector through which vertex has been translated.
Line	`int isUpdated`	F_{Lu}	Line update flag, see Table 7.2.
Line	`float translation[3]`	T_L	Vector through which centre of curvature has been translated.
Patch	`int isUpdated`	F_{Pu}	Patch update flag, see Table 7.2.
Element	`int isUpdated`	F_{Eu}	Element update flag, see Table 7.2.
Node	`int isUpdated`	F_{Nu}	Node update flag, see Table 7.2.
Node	`float translation[3]`	T_N	Vector through which node has been translated.

Table 7.2: Update flags.

Class	No change	Translated	Distorted		Notes
			in plane	out of plane	
Vertex	-	-	-	-	Store T_V
Line (F_{Lu})	0	1	2	3	Store T_L
Patch (F_{Pu})	0	1	2	3	
Node (F_{Nu})	0	1	-	-	Store T_N
Element (F_{Eu})	0	1	2	2	

7.3 Re-meshing scheme

7.3.1 Introduction

When a geometric change is applied to a model, the mesh must be updated. This is provoked by some dynamic cursor operation which is expected to be of pixel order. The mesh updating procedure is designed to minimise the number of elements affected when the model geometry is updated. This minimises the number of integrals, contained in the system matrix, that need to be recalculated before re-analysis and improves the convergence rate of a preconditioned iterative solver [17]. If the same total number of elements is maintained then the preconditioning matrix used by the solver can be re-used. If not then the matrix will need to be expanded. This could be done through block partitioning such as that described by Wang *et al.* [10]. For the purposes of this work it has been assumed that the number of elements is constant. However the method could be expanded or improved by adding or removing elements.

A flowchart summarising the general meshing strategy used in this work is given in Figure 7.3. The strategy is covered in more detail in the following sections. All updates are initially applied at the patch level; lines and vertices are considered if the algorithm needs more detail.

7.3.2 Translation

A patch is translated if all the features on the patch are modified through pure translation through the same vector. An example of this is the internal surface of a hole being moved within a plate of constant thickness. If a patch is translated, all the nodes and elements on the patch are moved consistently with the patch.

7.3.3 Distortion in plane

Distortion is caused by the differential movement of any two features included in a single patch such as a hole being moved within a patch. Two forms of distortion exist and require different approaches to

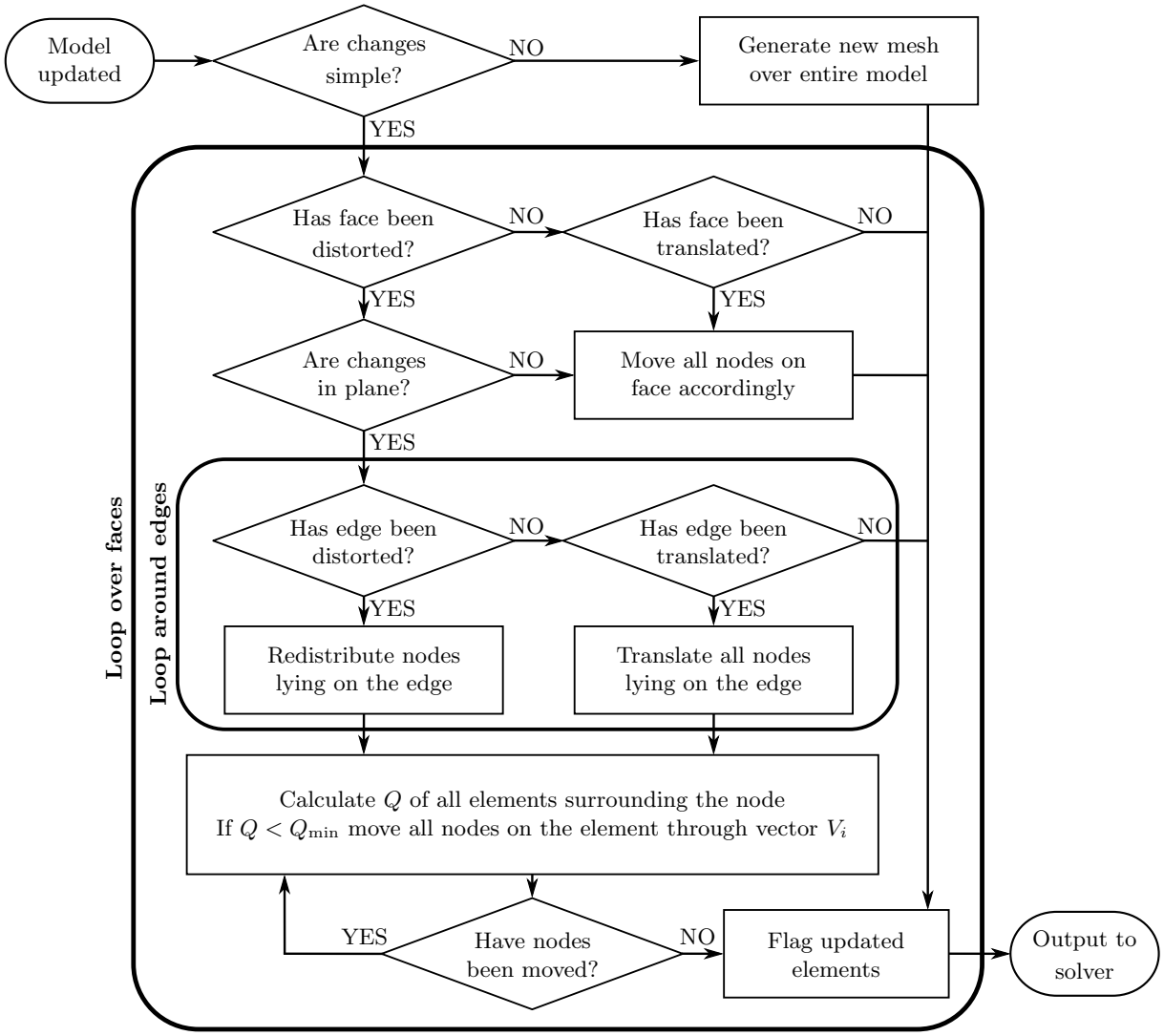


Figure 7.3: Flowchart showing a single re-meshing iteration.

update the mesh. The patch may distort out of plane, therefore requiring all the elements to be updated, or remain in plane, leaving some elements unaffected. If the patch remains in plane, a minimal number of elements should be updated without significantly reducing the mesh quality.

In-plane updating begins at the edges of the patch. Nodes on the lines around the patch are updated first. These may be translated if the line is translated or completely recalculated if the line has changed axis. However, if the axis of the line remains the same with either vertex at the same coordinates, the changes are propagated from the end of the line that has moved. Spacing between nodes is decreased (or increased) until the nominal minimum or maximum segment size is reached, starting from the end of the line which has changed, as shown in Figure 7.4.

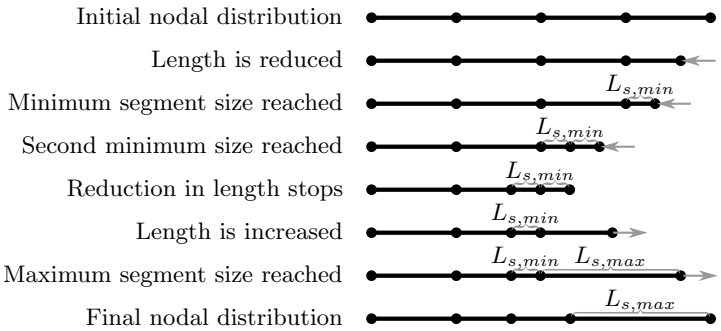


Figure 7.4: Nodal redistribution as the length of a line is changed.

If the axis of the line has changed, the segments along the line are scaled to preserve the nodal distribution. The scaling method for a straight line is described by:

$$N_i' = V_1' + l_i' = V_1' + (V_2' - V_1')\frac{\|N_i - V_1\|}{\|V_2 - V_1\|} \tag{7.1}$$

where the variables are defined as shown on Figure 7.5. All the variables are coordinate vectors; l_i is the vector along the line to the current node, N_i. During the geometric update of the problem the old locations of V_1 and V_2 are overwritten with the new values. Equation (7.1) must therefore be re-written:

$$N_i' = V_1' + (V_2' - V_1')\frac{\|N_i - (V_1' - T_1)\|}{\|(V_2' - T_2) - (V_1' - T_1)\|} \tag{7.2}$$

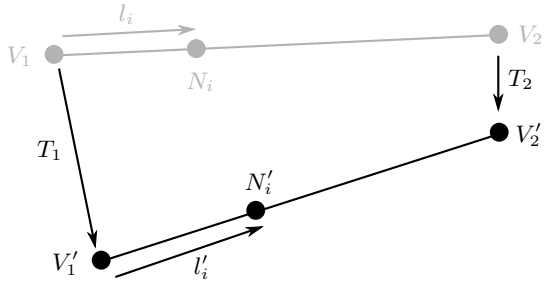


Figure 7.5: Scaling nodes on a straight line.

The variables for circular arcs, which are treated in a similar manner, are defined in Figure 7.6. The transformation is given by:

$$N_i' = C' + r_i' = C' + R((V_1' - T_1) - (C' - T_c))\|r_i'\| \tag{7.3}$$

where $\|r_i'\|$ is the new radius, R is the rotation matrix about the axis of rotation, a, of the arc:

$$R = \begin{bmatrix} a_x^2 + (1 - a_x^2)\cos\theta_i' & a_x a_y(1 - \cos\theta_i') - a_z \sin\theta_i' & a_x a_z(1 - \cos\theta_i') + a_y \sin\theta_i' \\ a_x a_y(1 - \cos\theta_i') + a_z \sin\theta_i' & a_y^2 + (1 - a_y^2)\cos\theta_i' & a_y a_z(1 - \cos\theta_i') - a_x \sin\theta_i' \\ a_x a_z(1 - \cos\theta_i') - a_y \sin\theta_i' & a_y a_z(1 - \cos\theta_i') + a_x \sin\theta_i' & a_z^2 + \cos\theta_i' \end{bmatrix} \tag{7.4}$$

and θ_i' is given by:

$$\theta_i' = \theta_i \frac{\theta'}{\theta} \tag{7.5}$$

where

$$\theta = \tan^{-1}\left(\frac{a\cdot((V_1-C)\times(V_2-C))}{(V_1-C)\cdot(V_2-C)}\right) = \tan^{-1}\left(\frac{a\cdot((V_1'-C'-T_1+T_c)\times(V_2'-C'-T_2+T_c))}{(V_1'-C'-T_1+T_c)\cdot(V_2'-C'-T_2+T_c)}\right) \tag{7.6}$$

$$\theta_i = \tan^{-1}\left(\frac{a\cdot((V_1-C)\times(N_i-C))}{(V_1-C)\cdot(N_i-C)}\right) = \tan^{-1}\left(\frac{a\cdot((V_1'-C'-T_1+T_c)\times(N_i-C'+T_c))}{(V_1'-C'-T_1+T_c)\cdot(N_i-C'+T_c)}\right) \tag{7.7}$$

and

$$\theta' = \tan^{-1}\left(\frac{a\cdot((V_1'-C')\times(V_2'-C'))}{(V_1'-C')\cdot(V_2'-C')}\right) \tag{7.8}$$

The $\tan^{-1}$ formulation is used to give an angle in the range $0 < \theta \leq 2\pi$. A simpler formulation could be implemented using $\sin^{-1}$, however this would only give the angle in the range $0 < \theta \leq \pi$ which is not sufficient for this application.

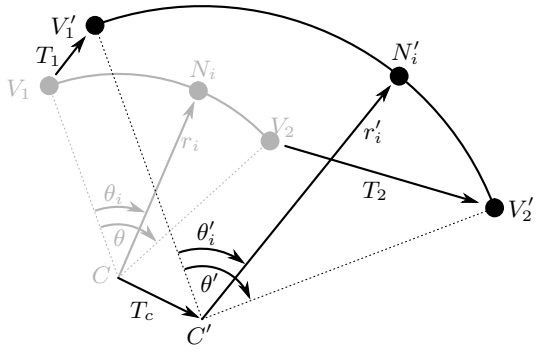


Figure 7.6: Scaling nodes on a circular arc.

Any patches that feature a structured mesh are reconstructed using the new nodal positions along the bounding lines to interpolate elements across the patch. Triangular meshed patches are more complex. In Figure 7.7, nodes $N_1 - N_4$ lie on the boundary of a circular hole that is moving across a patch. The vector through which N_i has moved, V_i, is stored. Each element that includes N_i, E_{ei} ($e = 1, 2, 3, ..., n_{Ei}$), is assessed for quality using equation (6.10) which is reproduced here:

$$Q = \frac{16A^2}{L_aL_bL_c(L_a+L_b+L_c)} \tag{7.9}$$

where A is the area of element E_{ei} and L_a, L_b, L_c are the side lengths of element E_{ei}.

The nominal minimum permitted element quality is designated $Q_{\min}$. If $Q > Q_{\min}$ then no further nodes are moved and only elements $E_1 - E_5$ are updated. However, if $Q < Q_{\min}$, the un-updated corner node(s) of E_{ei} are moved through vector γV_i, where $\gamma \geq 1$. The coefficient γ is chosen such that the mesh changes will not propagate beyond this node for several future updates. A node may be moved only once each time the model is re-meshed. Once all elements return an acceptable Q, propagation of the mesh updates ceases. All the updates act only on the corner nodes around each element; once all the required updates have been carried out, the mid-side node positions for updated elements are recalculated.

Figure 7.8 shows that elements exist in a series of concentric bands around the hole. The same banding can be observed in almost all automatically generated meshes (finite element (FE) or boundary element (BE)) around any geometric feature. Small perturbations in the location of the feature will affect only the neighbouring band. If the changes are larger, a cluster of elements of quality $Q_{\min}$ that move with the updated geometric feature will be maintained. $Q_{\min}$ may be graded by band to maintain a high element quality immediately around key geometry, leading to a more accurate approximation to the

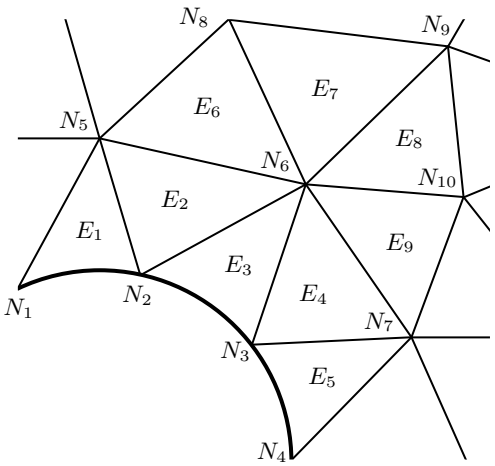


Figure 7.7: Element updating as a hole is moved.

stress in these areas. However, maintaining a high element quality results in changes propagating further through the mesh and the usual compromise must be made between speed and accuracy. It should be noted that, if the direction in which the feature is being moved reverses, the elements of $Q_{\min}$ quality will be distorted in such a way that Q will initially increase across these elements.

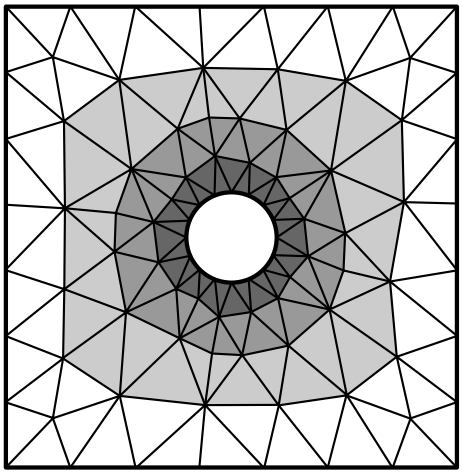
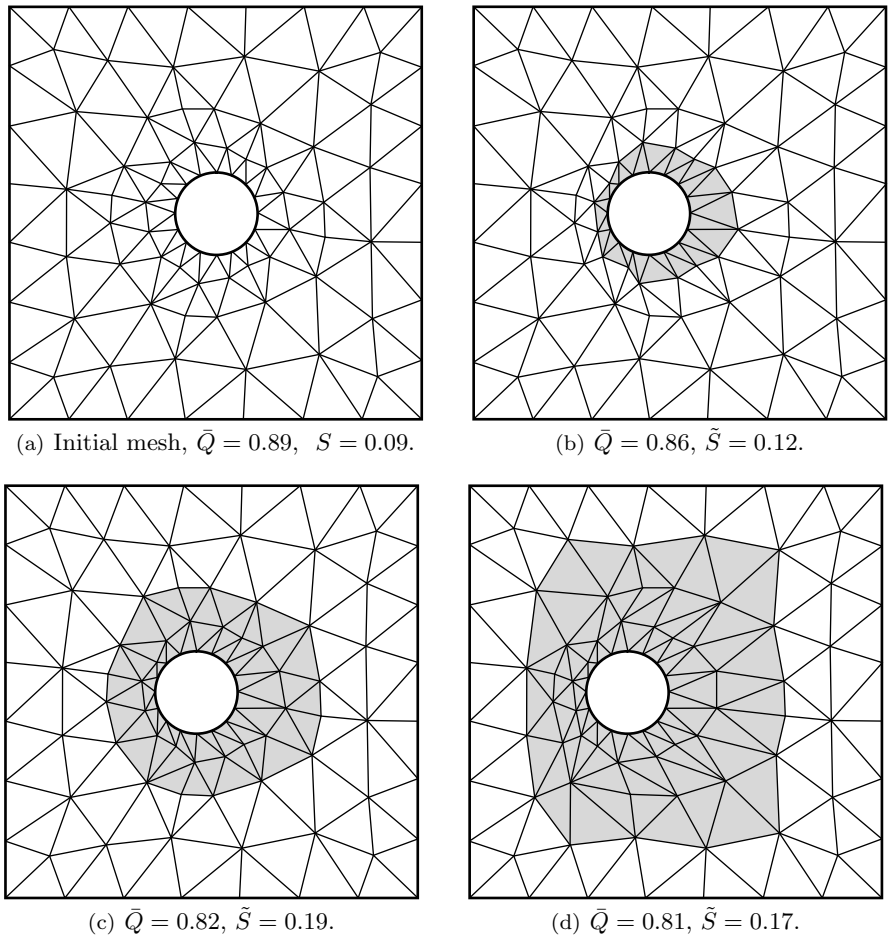


Figure 7.8: Element banding around a hole.

The changes applied to the mesh generally result in increasing numbers of updated and distorted elements and hence degradation of the mean element quality, $\bar{Q}$, as demonstrated in Figure 7.9, which illustrates how the updates propagate across a patch as a circular hole through it is iteratively moved to the left. In this example, $Q_{\min}$ has been set to 0.5. A high quality initial mesh absorbs some of the distortion but, to preserve accuracy, the mesh will periodically require more extensive modification over distorted patches. For large distortions, the mesh must be regenerated across the entire patch (this rule has not been applied in Figure 7.9 so that the effects of failing to apply this measure can be seen). Once the updates have finished propagating, the mean and standard deviation of the element quality across patch j, $\bar{Q}_j$ and $\tilde{S}_j$, are calculated. If $\bar{Q}_j < \bar{Q}_{\min}$ or $\tilde{S}_j > \tilde{S}_{\max}$, the mesh across patch j is regenerated when the re-meshing algorithm is next called, in place of the usual nodal updating procedure. This is done by following the steps in Section 7.3.4. For smaller problems, elements may be removed from areas where the mesh has become dense, such as in front of an advancing hole, and replaced in sparse areas,

such as behind the hole as shown in Figure 7.10. This will result in modifications being made to a smaller number of elements than re-generating the entire mesh, however $\bar{Q}_j$ will not be as substantially improved. Both techniques will result in an increase in run time for analyses where a part of the mesh has been regenerated but effectively reduce it on the following runs.

The ratio of elements updated to total elements on the patch, n_{Eu}/n_E, may also be used as a trigger for re-meshing. If a large number of elements are being repeatedly updated in a series of iterations it may prove faster to expend some additional time in a single iteration to re-mesh the patch. The subsequent iterations will thereby affect a smaller number of elements and therefore be faster to execute.



(a) Initial mesh, $\bar{Q} = 0.89$, $S = 0.09$.

(b) $\bar{Q} = 0.86$, $\tilde{S} = 0.12$.

(c) $\bar{Q} = 0.82$, $\tilde{S} = 0.19$.

(d) $\bar{Q} = 0.81$, $\tilde{S} = 0.17$.

Figure 7.9: Iterative updating of the mesh around a hole, showing updated elements. $Q_{\min} = 0.5$.

The reader is reminded that, since the mesh is updated in real-time as the user drags features of the object into new locations, the majority of the geometric changes will be of pixel order and will require only minor modifications to the mesh. The larger mesh updates will be applied only in a very small proportion of the calls to the algorithm.

7.3.4 Distortion out of plane

Patches distorted out of plane will require the entire mesh across the patch to be updated. First, the stored transformation data for the new patch in the two-dimensional, (u, v) domain, is updated. The location of the nodes along the lines are scaled using the method as described in Section 7.3.3 before regenerating the mesh following the same procedure as was used to generate the initial mesh. The same numbering system for the nodes around the patch boundary will be maintained but all internal nodes

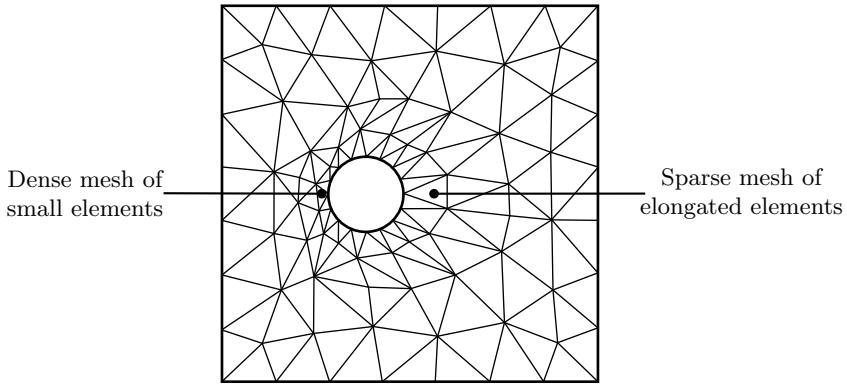
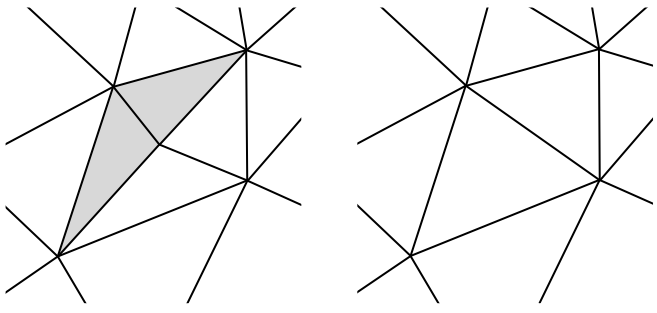


Figure 7.10: Exaggerated density distribution of updated mesh as a hole is moved to the left.

across the patch will be replaced.

The number of elements in the new mesh should match the number in the old mesh. This preserves the properties of the analysis matrix. Rows and columns relating to the current patch remain in the same place and do not affect the location of rows and columns relating to other patches. This is important as it means the same preconditioning matrix can be applied to the updated model. This preconditioner approximates the inverse of the system matrix and pre-multiplies both sides of the linear system of equations produced by the boundary element method (BEM). This improves the structure of the system matrix by moving it closer to the identity matrix, making it faster to solve. Once the new mesh has been generated, an algorithm checks the number of internal nodes on the patch and adds or removes them as necessary. The number of nodes is directly related to the number of elements. If a node is added or removed a pair of elements will be added to or removed from the node.

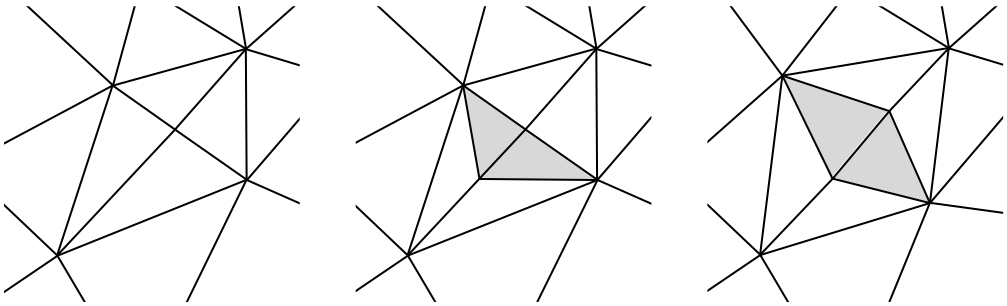
To reduce the number of elements on a patch they are removed from an element quad (four elements around a single node). Initially the central node and all the surrounding segments are removed. A new segment is then created across the shortest distance between two opposite bounding nodes thus defining the two updated elements. This process is illustrated in Figure 7.11. The remaining elements are removed from the data structure. If there are no element quads available then the algorithm returns an error message. Generally this is not a problem, however for robustness the algorithm needs to be developed to remove elements when there are no quads available. It is possible to collapse any pair of elements in the model so long as they share a common segment. It may be preferable to use an algorithm similar to that used to collapse poor quality elements and remove the worst quality elements from the patch. However, this is a more computationally expensive process.



(a) Original mesh. (b) After removal.

Figure 7.11: Removing a pair of elements.

Elements can be added by creating a new node at the centre of the longest segment on the patch. New segments are added by dividing the neighbouring elements into two, thus creating two new elements. This process is illustrated in Figure 7.12. If elements have been added or removed whilst regenerating the mesh across the entire patch a Laplacian smoothing routine is applied across the entire patch, otherwise it is applied locally as shown in Figure 7.12(c).



(a) Original mesh. (b) After addition. (c) After smoothing.

Figure 7.12: Adding a pair of elements.

7.4 Validation

7.4.1 Test Model

Tests have been carried out to validate the accuracy of stress solutions calculated using BE models perturbed by the re-meshing scheme. One example is discussed here, the case of moving a circular hole, of diameter D, within a thick plate of dimensions $15D \times 5D \times D$. The stresses at the nodes around the hole were computed using a BE mesh produced using the new algorithm and compared to benchmark stresses produced using a converged FE model. The FE model was converged to a L_2-norm tolerance of less than 1% in the stresses at the nodes around the hole. A long plate with a hole has been chosen to reduce the effects that the proximity of the ends of the plate have on the stresses around the hole. A single benchmark can therefore be used to compare the stresses for all hole locations as it is moved along the plate. The BE mesh on the surfaces $y = 0$ and $z = D$ of the model is shown in Figure 7.13. The following boundary conditions are applied:

$$\begin{cases} u_x = 0, & x = 0 \\ u_y = 0, & y = 0 \\ u_z = 0, & z = 0 \\ t_x = \tilde{t}, & x = 15D \end{cases} \tag{7.10}$$

where u and t refer to displacements and tractions applied in the subscripted Cartesian direction. We take $\tilde{t} = 1000$. To validate the mesh, 13 Gauss points are used to integrate all triangular elements and 16 to integrate quadrilateral elements.

An error measure, ε_σ, is calculated to compare the accuracy of the BE stresses to the benchmark:

$$\varepsilon_\sigma = \frac{\|\{\sigma\} - \{\hat{\sigma}\}\|}{\|\{\hat{\sigma}\}\|} \tag{7.11}$$

where $\|\cdot\|$ denotes the L_2-norm, $\{\hat{\sigma}\}$ the benchmark stress components and $\{\sigma\}$ the matching stress components generated by the BE code. For this study, the tangential stresses, $\{\sigma_t\}$, around the top of

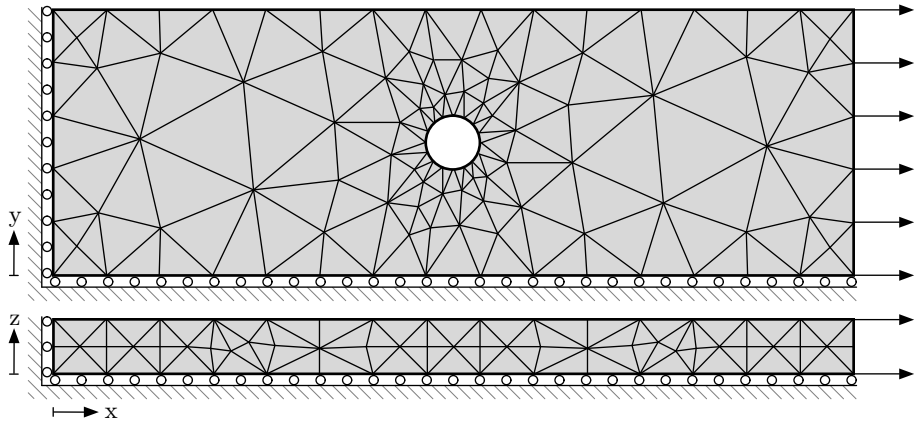


Figure 7.13: Test model.

the hole ($z = D$) will be used.

7.4.2 Test results

Figure 7.14 shows the normalised tangential stress, $\sigma_t/\tilde{t}$, after the hole has moved a distance, $d = 0.5D$ in the positive x direction. Angle θ is defined as the anticlockwise angle from the top of the hole, as viewed in Figure 7.13. Note that, as this is a thick plate, the stress concentration factor, K_t, is smaller on the free surface of the plate than at $z = 0$ where $K_t \approx 3.1$ consistent with expectations for thin plate theory [2].

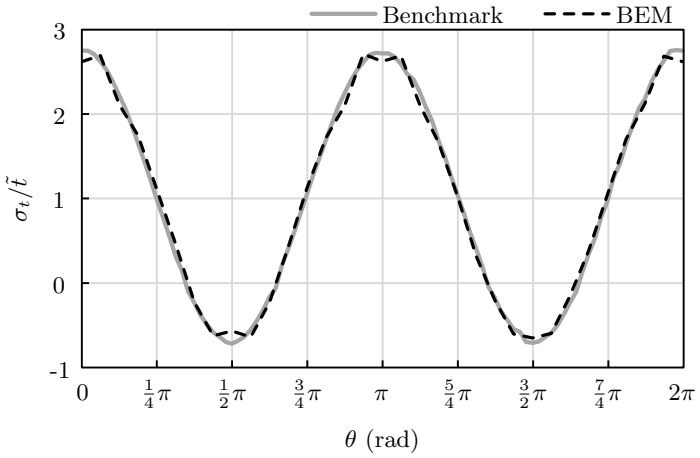
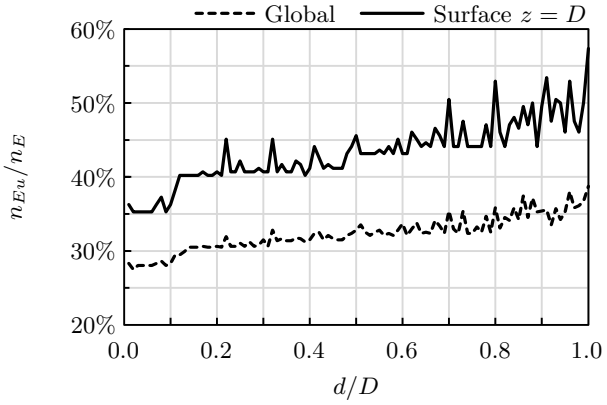


Figure 7.14: Normalised tangential stress, $\sigma_t/\tilde{t}$, around hole after it has moved through distance $D/2$.

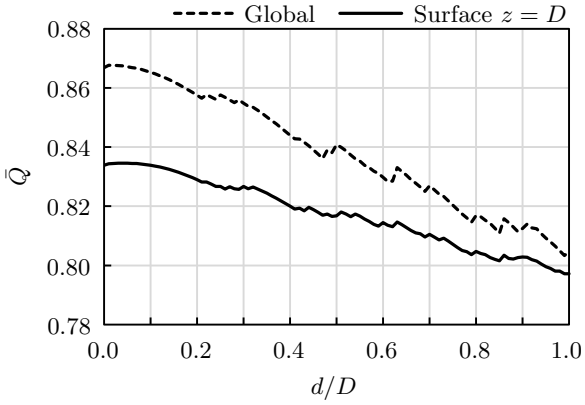
The deviation in the BE results from the benchmark solution is most noticeable when $d\sigma_t/d\theta = 0$. Similar deviations also appear in a given FE model if a comparable number of elements is used around the hole. All the elements around the hole were initially slightly distorted, having $\bar{Q} = 0.77$. However, at around $\theta = 3\pi/2$ radians; deformation of these elements as the mesh was updated resulted in a local increase in $\bar{Q}$ and hence produced a better approximation to $\hat{\sigma}_t$ when compared to $\theta = \pi/2$ where $\bar{Q}$ has degraded locally. A smoother and more accurate curve can be produced through appropriate application of higher quality elements; however, the trade-off between accuracy and computational resources required must be considered. The emphasis in the current work is to achieve very rapid solutions of acceptable quality during interactive re-analysis; higher quality solutions can be obtained using a more refined mesh. However, this will require a more computationally expensive analysis and should only be carried out to

assess the final stress values.

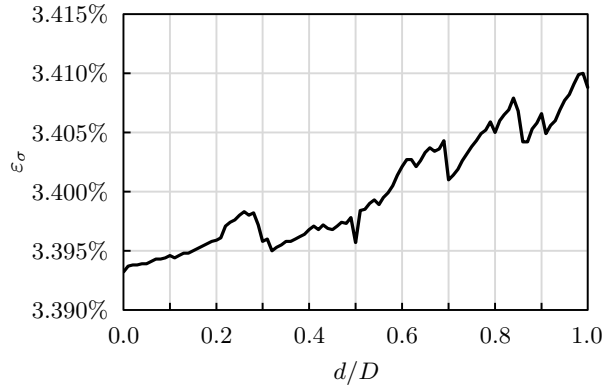
A profile of how the model behaves as it is modified has been produced for the plate with a hole when it is meshed with 700 elements. The hole was moved through distance D in the positive x direction in 100 steps to simulate pixel order updates. The results are shown in Figure 7.15 where d is the total distance through which the hole has moved at any given point. It can clearly be seen in Figure 7.15(a) that if the hole is continuously moved in the same direction the number of updated elements, n_{Eu} will generally increase. To reduce this accumulation, local mesh regeneration must be carried out periodically. Without these schemes, $\bar{Q}$ will, on average, decrease as the hole is moved further, as shown in Figure 7.15(b), leading to an increase in ε_σ, as shown in Figure 7.15(c). To show the effects of periodically regenerating the mesh, an arbitrary value of $\bar{Q}_{\min} = 0.84$ is selected. The points at which the mesh is regenerated are shown by the grey lines in Figure 7.15(d). It can be seen that some of the element distortion is removed and $\bar{Q}$ is effectively regulated. This in turn helps to regulate ε_σ. However, it is clear from Figure 7.15(c), which shows ε_σ without any regeneration, that $\Delta\varepsilon_\sigma$ remains small during re-meshing relative to ε_σ, accommodating some element distortion as evident in Figure 7.15(b). It should be noted that ε_σ is the L_2-norm error; the error in the peak stress is typically around $\varepsilon_\sigma/2$. Wall clock timings give the times for re-meshing in the region of thousandths of a second, around six times faster than generating the initial mesh.



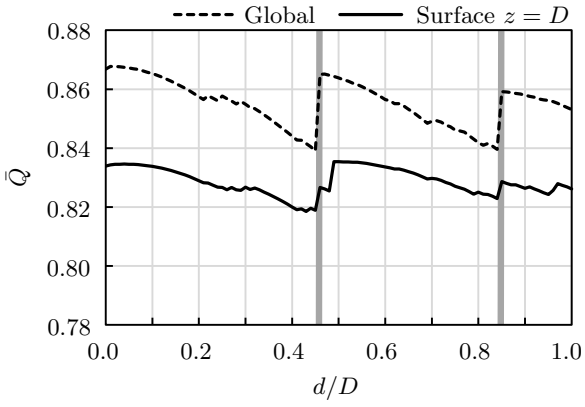
(a) Percentage of elements updated for a plate with a hole.



(b) Mean element quality for a plate with a hole.



(c) Error, ε_σ, at nodes around hole.



(d) Mean element quality with regeneration.

Figure 7.15: Re-meshing validation.

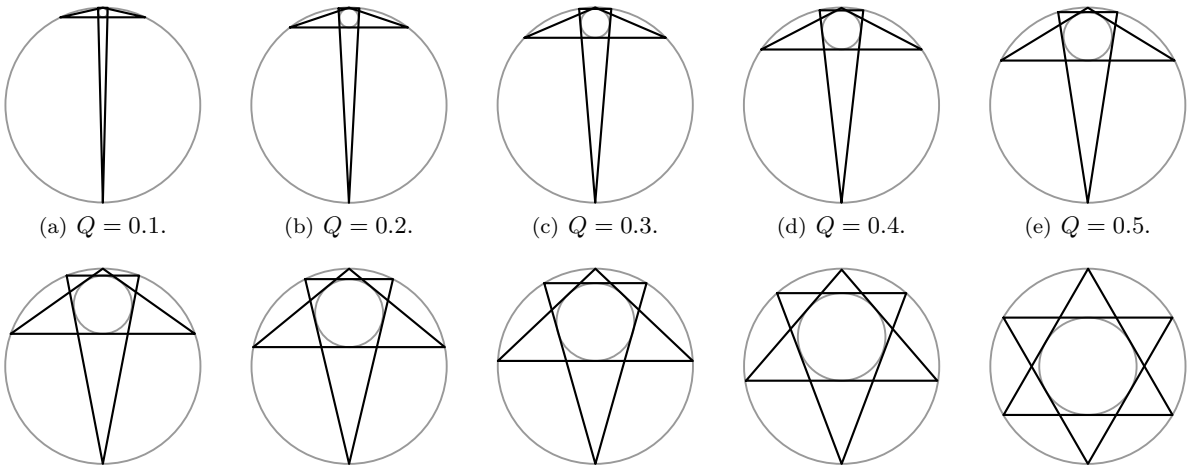
If applied intelligently a small but sufficient number of high quality elements can produce a more accurate solution than many low quality elements. For the fastest analysis, it is therefore essential to find and generate an optimal mesh. Generally stresses peak on the boundary of a model. If a high quality

local mesh is maintained in these areas a greater accuracy can be achieved and maintained for these key results. However, this causes a greater number of elements to be updated in each iteration as the model is deformed and hence will increase the analysis time. A compromise must be reached to enable real-time updating whilst maintaining an acceptable accuracy.

7.5 Selection of the nominal minimum element quality

A nominal value of $Q_{\min} = 0.6$ was used to generate all of the results in Section 7.4. In this section, different values of $Q_{\min}$ are assessed to find the best compromise between $\bar{Q}$, n_{Eu}/n_E and ε_σ. It should be noted that $Q_{\min}$ is not the lowest element quality in the model, rather it is the threshold where elements of $Q < Q_{\min}$ are updated by the re-meshing algorithm.

Figure 7.16 shows Q, determined using the circle ratio, for several isosceles triangular elements. Scalene elements could take many more forms. A purely qualitative approach suggests that $Q_{\min} = 0.6$ is a sensible choice. However, full scientific rigour should be applied to establish if, in fact, this is the case. Therefore a set of models was constructed using different values of $Q_{\min}$ to assess this measure. The results of these tests are summarised in Figure 7.17. It should be noted that the mesh was not completely re-generated at any point during the tests.



(a) $Q = 0.1$. (b) $Q = 0.2$. (c) $Q = 0.3$. (d) $Q = 0.4$. (e) $Q = 0.5$.

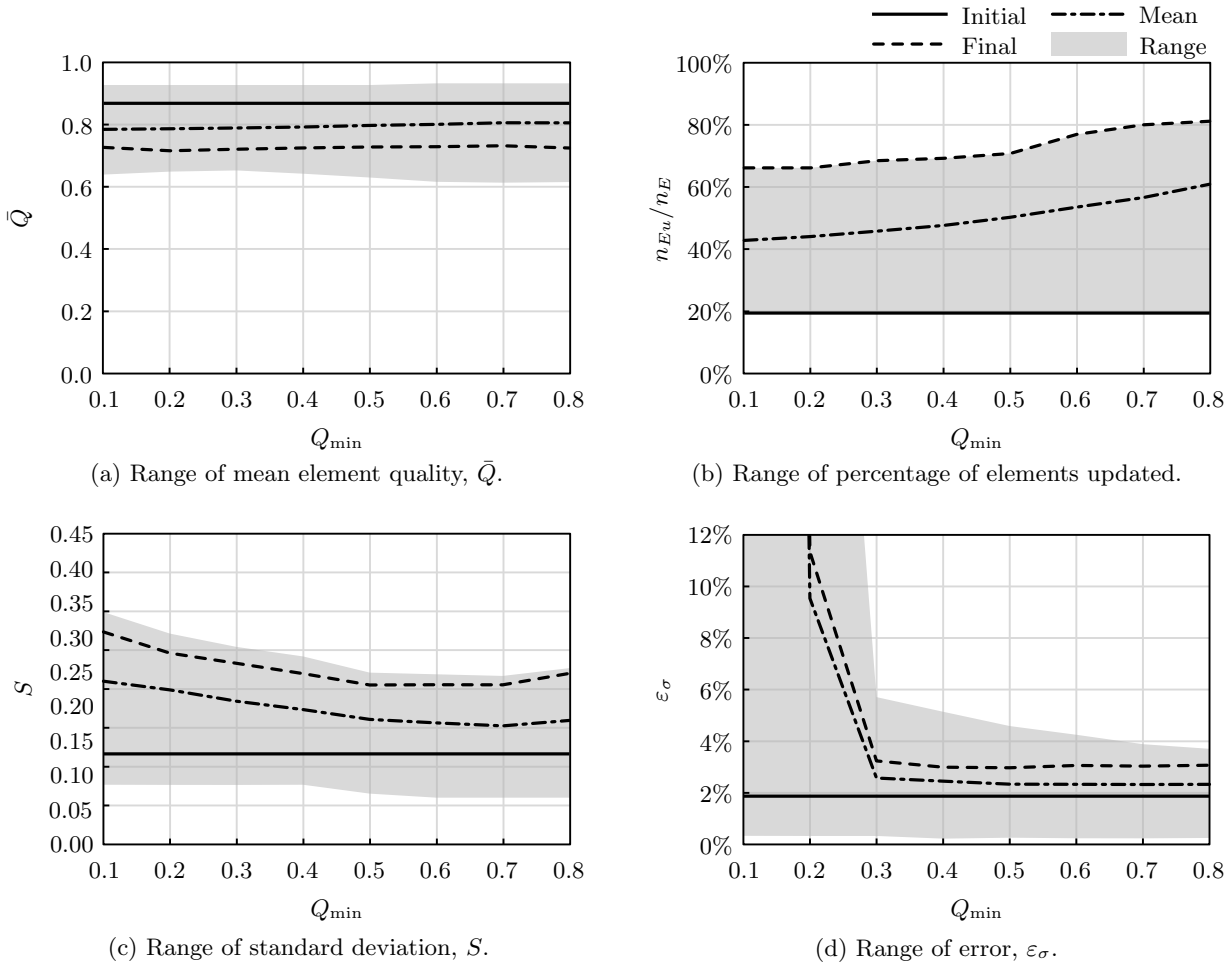
(f) $Q = 0.6$. (g) $Q = 0.7$. (h) $Q = 0.8$. (i) $Q = 0.9$. (j) $Q = 1.0$.

Figure 7.16: Distortion of isosceles elements.

Figure 7.17(a) shows that the mean element quality stays approximately constant whatever value of $Q_{\min}$ is used. This is because, when $Q_{\min}$ is small, very few elements in the mesh are updated so that the remainder of the mesh retains the high quality it exhibited after initial generation. When $Q_{\min}$ is large, maintaining high quality updated elements means that the updates to the mesh propagate to the edges of each patch. This leads to poor quality elements around the edges as the nodes on the edges cannot be translated using the normal scheme without moving them outside the boundaries of the model.

The percentage of elements updated, n_{Eu}/n_E, increases steadily with $Q_{\min}$, as shown in Figure 7.17(b). This is to be expected as maintaining a higher element quality requires that the changes to the mesh propagate further. The smaller $Q_{\min}$, the faster the re-analysis. Therefore it is desirable to select the smallest possible $Q_{\min}$ which gives results to an acceptable accuracy. Figure 7.17(d) suggests that this occurs when $Q_{\min} = 0.3$.

The accuracy of the solution can be improved by maintaining a high quality band of elements around key geometric features such as a hole in a plate. This is shown by the solid line in Figure 7.18. If the Q



(a) Range of mean element quality, $\bar{Q}$.

(b) Range of percentage of elements updated.

(c) Range of standard deviation, S.

(d) Range of error, ε_σ.

Figure 7.17: Collated effect of changing $Q_{\min}$ for a range of models.

immediately adjacent to the hole is allowed to degrade, ε_σ increases until it reaches an equilibrium when $Q = Q_{\min}$ and will therefore not degrade further. This is shown by the dashed line.

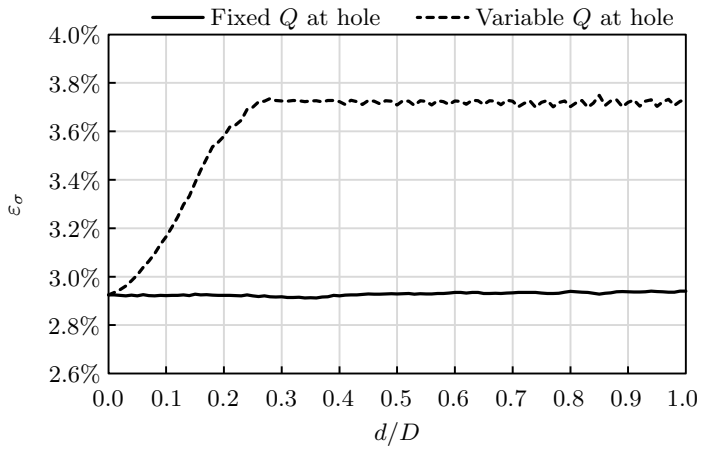


Figure 7.18: Error when using a high quality band of elements around a hole.

Through analysis of the full range of results which were summarised in Figure 7.17, the author suggests $Q_{\min} = 0.4$ as a suitable compromise that maintains an acceptable error, ε_σ, whilst limiting the number of updated elements and hence the re-analysis time.

Chapter 8

Acceleration of the integration phase

"No idea is so outlandish that it should not be considered with a searching, but at the same time, I hope, with a steady eye."

Winston Churchill

8.1 Introduction

This chapter describes the algorithms and computational techniques used to accelerate the re-integration of the boundary integral equation (BIE) and the re-construction of the boundary element (BE) system of equations. A detailed analysis is carried out to establish which modifications to the standard algorithm result in speed gain whilst maintaining an acceptable degree of accuracy.

8.2 Reconstruction of the system

8.2.1 Re-integration strategy

The re-meshing algorithm used in this work aims to limit propagation of geometric changes through the mesh to those areas close to the updated geometry. This means that large parts of the mathematical system are unchanged. Re-integration of the system can therefore be accelerated by re-using any unchanged $[H]$ and $[G]$ sub-matrices. This is done by following the flowchart given in Figure 8.1.

Each updated element may be treated in one of two different ways: it may be translated or distorted. Updates to nodes may be thought of in the same manner: Translated nodes are those that lie on translated elements; distorted nodes lie on distorted elements. Where a node is shared by both translated and distorted elements, it is classed as translated. Where a paired element and node have the same translation applied, the associated $[H]$ and $[G]$ sub-matrices will not change. Any element or node classified as distorted will require recalculation of all the associated geometric data and sub-matrices. At present rotated elements are classed as distorted. However, if a group of elements are rotated together, the associated $[H]$ and $[G]$ sub-matrices will not change, as with the translation case.

It should be noted that the outer pair of loops in Figure 8.1 are interchangeable. The integration may be carried out by looping primarily over the field elements or the source nodes.

8.2.2 RISP integration scheme

In the current work a reusable intrinsic sample point (RISP) integration scheme is applied [29]. The RISP algorithm uses a discrete number of integration schemes, therefore the shape functions and derivatives for each integration scheme need only be calculated once for each element type.

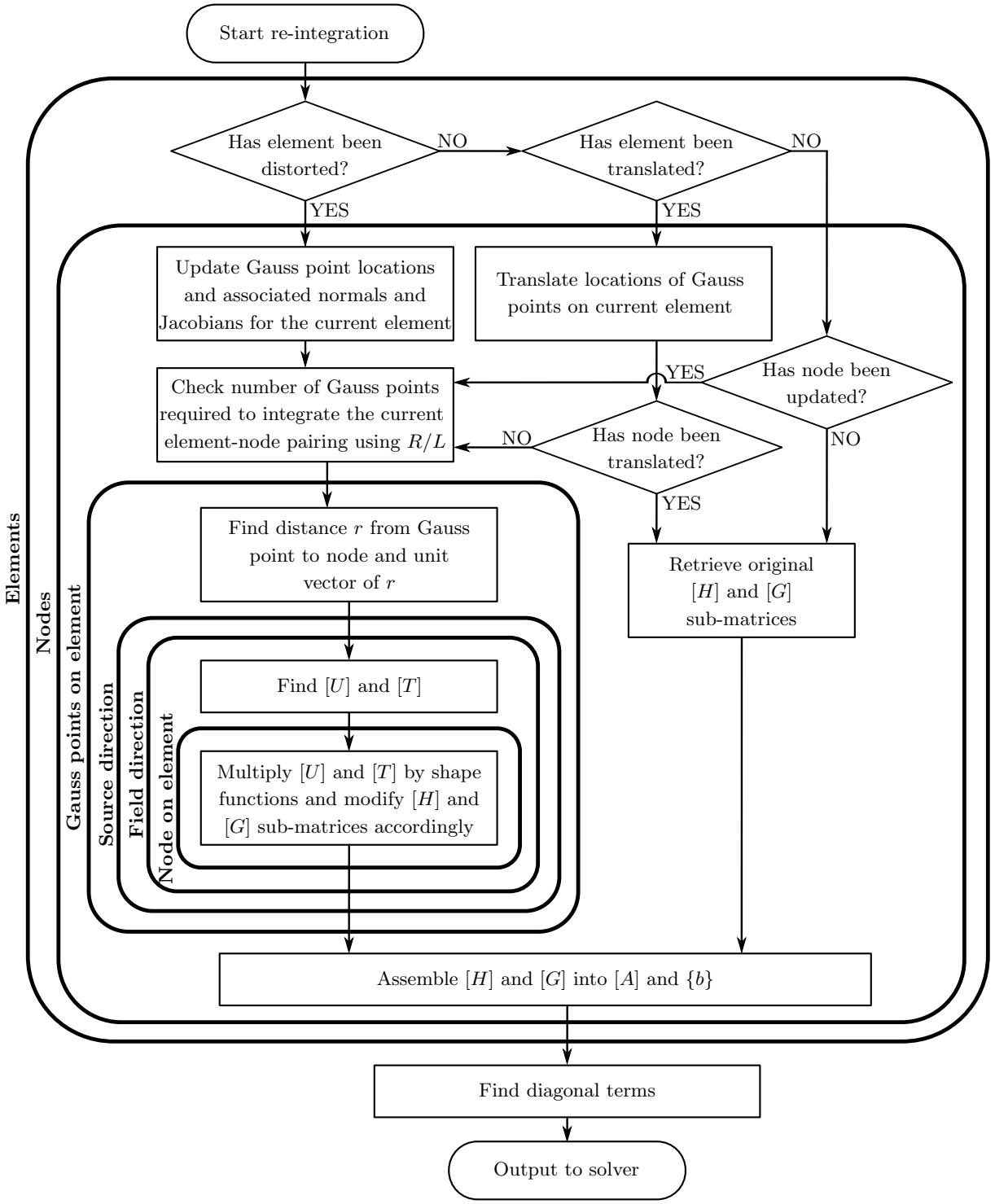


Figure 8.1: Flowchart of the re-integration algorithm.

In the current work, R is taken to be the distance from the centroid of the field element to the source node and L as the average side length of the element. Equations (4.21) and (4.23) can be rearranged to find the maximum ratio R/L for a given number of Gauss points. From (4.21):

$$\frac{R}{L} = \frac{1}{4}\left(\frac{e}{2}\right)^{-\frac{p'}{2m}} \tag{8.1}$$

From (4.23):

$$\frac{R}{L} = \frac{8}{3}\left(\frac{-10m}{p' - \ln(e/2)} - 1\right)^{-\frac{4}{3}} \tag{8.2}$$

This ratio can be used in conjunction with the RISP integration scheme to directly determine the appropriate number of Gauss points to use across the entire element. If e is taken to be 10^{-5}, Table 8.1 can be constructed, where $\hat{m}$ is the total number of Gauss points on the element. This scheme can now be applied to both quadrilateral and triangular elements. If the source node lies on the field element the near-singular integration scheme suggested by Kane [29] is applied.

Table 8.1: Total number of Gauss points, $\hat{m}$, for $e = 0.001$.

Condition		$\hat{m}$	
Lachat and Watson	Bu and Davies	Tri	Quad
Singularity at corner node		32	64
Singularity at mid-side node		36	96
$R/L < 1.6$	$R/L < 1.9$	13	16
$1.6 \leq R/L < 3.3$	$1.9 \leq R/L < 3.7$	7	9
$3.3 \leq R/L < 17.5$	$3.7 \leq R/L < 13.9$	4	4
$17.5 \leq R/L$	$13.9 \leq R/L$	1	1

Using a discrete number of integration schemes leads to a reduction in memory requirements and, as the models encountered in this work generally contain fewer than 2000 elements, it is possible to store all the geometric data associated with the integration in memory. The Gauss point locations, associated normals and Jacobians for all integration schemes are therefore only computed once for each element in the model. This data is stored so that, where possible, it can be re-used in the re-analysis. The near-singular integration schemes are formulated such that they can be applied using the same algorithm as the standard integrals.

8.2.3 Refining poor quality elements

An additional refinement of the integration scheme is introduced, whereby elements of poor quality would be re-integrated using a higher order integration scheme to reduce the potential error associated with the distortion. This scheme will be applied according to Table 8.2, where the divisions in Q have been spaced evenly starting from $Q_{\min} + 0.1$.

Table 8.2: Integration refinement scheme for poor quality elements.

	Current $\hat{m}$ (Triangular/Quadrilateral)			
	1/1	4/4	7/9	13/16
$Q > 0.4$	1/1	4/4	7/9	13/16
$0.4 \geq Q > 0.3$	4/4	7/9	13/16	13/16
$0.3 \geq Q > 0.2$	7/9	13/16	13/16	13/16
$0.2 \geq Q$	13/16	13/16	13/16	13/16

Integrals that multiply zero boundary conditions may be suppressed to remove the need to calculate these values.

8.3 Computational strategy

This section covers areas of the implementation that are not affected by the algorithm structure but are influenced by the computer architecture.

8.3.1 Precision

Double precision computing allows more precise storage and computation of data. However, it requires twice as much memory and more processor time than single precision. It is generally accepted, and often assumed, that double precision is needed for accurate BE computation. However, it has not been possible to find research that validates this fact. For this reason both single and double precision implementations of the algorithm will be assessed in this work.

8.3.2 Architecture

In this work we use the term system architecture to refer to the width of the data path to the processor and hence the length of the word that can be used to address the computer memory. Longer words allow more memory locations to be addressed. There are two primary processor architectures in use: 64- and 32-bit. The 32-bit architecture only allows access to 4GB of computer memory; by utilising a 64-bit architecture this can be extended to 16EB (an exabyte (EB) is 10^{18} bytes). However, with 64-bit architecture the same data occupies more space in memory than with a 32-bit architecture due to the longer pointers and additional alignment padding. This can also have implications for efficient utilisation of the processor cache, which could require more memory read/writes. If more than 4GB of memory is required a 64-bit system must be used.

In this work both 64- and 32-bit implementations will be compared for models that are small enough to use a 32-bit architecture.

8.3.3 Parallelisation

The majority of the literature associated with parallelising the boundary element method (BEM) deals with applying the method across multiple distributed memory devices. Here we parallelise the method on a single shared memory machine. This is a simple process, aided by the way the integration algorithm described in this work has been constructed.

If the outer loop of the integration scheme is over the elements, as described in Figure 8.1, and the algorithm is parallelised by sending each element to a different processor, assembly of $[A]$ and $\{b\}$ must be carried out after the integration has been completed. As the full $[H]$ and $[G]$ matrices are being stored for re-use during each subsequent re-analysis this is a trivial matter. The assembly of $[A]$ and $\{b\}$ may be independently parallelised by generating each row of the system on a different processor. If the outer loop of the algorithm is over the nodes, the construction of $[A]$ and $\{b\}$ may be carried out inside the main loop.

8.4 Test models

To assess the speed and accuracy of the re-integration, three test models have been considered to cover a range of stress conditions:

1. Thick walled cylinder (TWC) with an internal pressure. The internal radius has been reduced as i increases, where i is the number of updates applied to the model.
2. Cantilever (CTL) under bending. The length of the cantilever has been extended as i increases.

3. Plate with hole (PWH) under uniaxial tension. This gives a more complex stress field than the uniform field found in the thick walled cylinder. The hole has been moved along the plate as i increases.

Each model has been analysed using several different meshes with varying degrees of refinement and different step sizes of geometric perturbation. The results given in the current chapter summarise the results of analysis carried out over all of these test cases.

The nomenclature used in the following sections will be as follows: the times, in seconds, to update $[A_i]$ is denoted t_{ui}. For the first iteration, $i = 0$, t_{u0} is always the time to fully populate the system. When $i > 0$, t_{ui} is the time required to update the system using the current scheme. The total number of degrees of freedom (DOF) is n and the number of updated DOF at the ith update is denoted n_{ui}. The total number of Gauss points across each element is denoted $\hat{m}$. The speed-up factor for each scheme, S, is calculated from $S = t_{u0}/t_{ui}$. An error measure, ε_x, is used to compare the accuracy of the scheme to the benchmark:

$$\varepsilon_x = \frac{\|\{x\} - \{\hat{x}\}\|}{\|\{\hat{x}\}\|} \tag{8.3}$$

where $\|\cdot\|$ denotes the L_2-norm, $\{\hat{x}\}$ the benchmark solution vector and $\{x\}$ the BEM approximation to $\{\hat{x}\}$.

With the exception of Section 8.5.1, the benchmark models to which each acceleration scheme has been independently applied use the re-integration algorithm described in Figure 8.1. This is applied with 13 Gauss points for all non-singular integrals across triangular elements and 16 Gauss points for all non-singular integrals across quadrilateral elements. For singular and near-singular integrals the scheme given in Section 4.2 is applied. The algorithm is implemented using double precision on a single processor with a 32-bit architecture. This set of conditions has been chosen so that ε_x reflects the increase in error of applying the chosen acceleration scheme.

In Section 8.5.1, the benchmark models use a refined version of each mesh where each element has been divided into four parts. This is used with $\hat{m} = 28$ for standard integrals and the normal singular integration scheme to ensure a high degree of accuracy in $\{\hat{x}\}$ so that the effect of reducing $\hat{m}$ can be accurately assessed.

8.5 Results

This section contains an overview of the tests carried out on the different analysis acceleration techniques. Each technique is applied individually to the basic re-integration scheme so that the merits of applying each scheme can be independently assessed. The most beneficial schemes are then combined for the final set of tests.

8.5.1 Re-integration

Figure 8.2 summarises the error, ε_x, if the same number of Gauss points, $\hat{m}$, is used to integrate over each element to evaluate all regular boundary integrals. In Figure 8.2, $\hat{m}$ refers to the number of Gauss points used to integrate over each triangular element.

Figure 8.2 shows that the error increases rapidly if fewer than 5 Gauss points are used across each element. Less than 0.3% reduction in the error can be achieved by using more than 7 Gauss points. As the system construction time increases with the number of Gauss points this indicates that, for maximum efficiency whilst retaining a suitable accuracy for a fixed integration scheme, the standard 7 Gauss point scheme should be applied to all triangular elements, this corresponds to a 9 point scheme across quadrilateral elements. The similarity between the shape of the curves in Figures 8.2 and 4.10 should be noted.

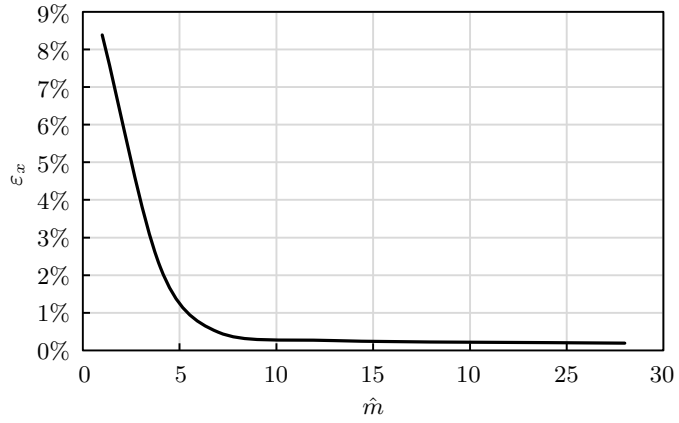


Figure 8.2: Accuracy using a fixed integration scheme across all elements.

If only the updated node-elements pairings are re-integrated during re-analysis the speed-up achieved is shown in Figure 8.3. Due to the overheads involved the speed-up factor, $S < 1$ if almost all of the elements in the model were updated. However, the re-meshing algorithm will not permit this many elements to be updated in one iteration.

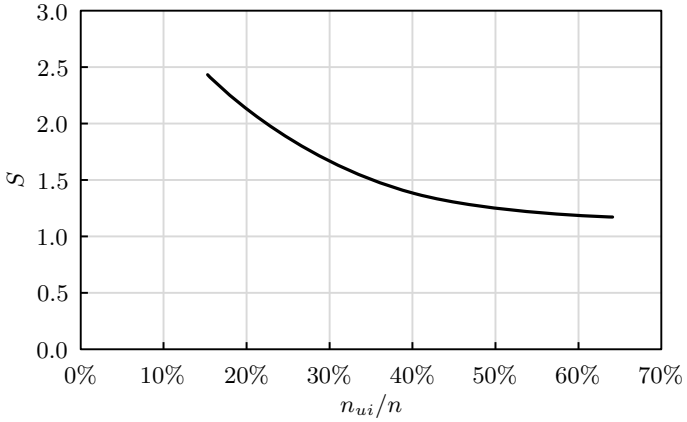


Figure 8.3: Acceleration achieved using partial re-integration.

8.5.2 Integration scheme

Figure 8.4 compares the speed-up, S, achieved using the Bu and Davies [122] and Lachat and Watson [119] integration schemes. The solid lines show the effect of updating the integration scheme, based on R/L, across each element. The dashed lines show the effect of re-applying the scheme calculated during the initial analysis. The dotted lines show the effect of refining the integration scheme based on both R/L and the quality, Q, of the element.

The Bu and Davies [122] scheme typically runs around 3% faster than the Lachat and Watson [119] scheme for problems where $n > 1000$ as it uses fewer Gauss points. This coarser scheme leads to a small increase of 0.01% in the error, ε_x. For problems where $n < 1000$ the Lachat and Watson scheme performs fastest. This is because R/L is generally small in small models and very few elements are integrated using a single integration point. The main speed advantage in Bu and Davies' scheme is gained when $R/L > 13.9$ and this is normally lost when $n < 1000$.

If the same integration scheme is used for each element-node pairing as was applied the initial analysis, an additional speed-up can be achieved. However, this is negligible and is at the expense of a much greater

variability in the error, ε_x, especially as the change to the geometry becomes large. For this reason $\hat{m}$ should be updated periodically, if not after every update iteration.

Using a higher order integration scheme over poor quality elements leads to a reduction in the error of less than 0.01%. This is at the expense of a 4% decrease in S.

The Bu and Davies scheme has been adopted for use in the integration algorithm. This has been implemented without adjustment for poor quality elements and $\hat{m}$ is updated during each update iteration. Using this scheme the accuracy remains comparable to using a 13 Gauss point scheme across every element whilst the solve time is reduced to that of using 4 Gauss points.

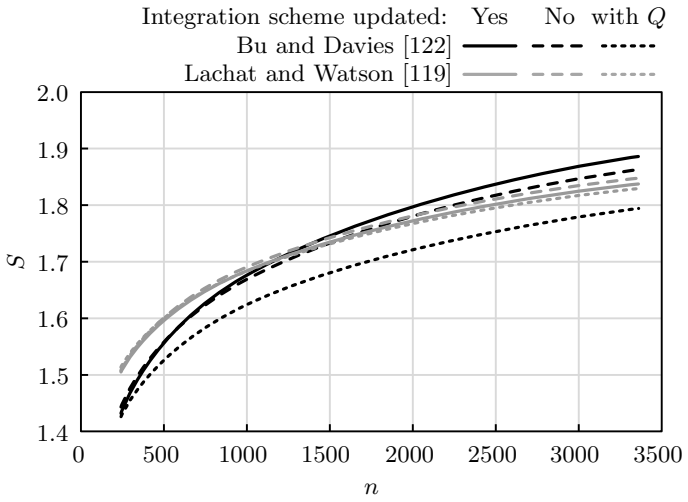


Figure 8.4: Speed up factor of variable integration schemes.

8.5.3 Suppression of integrals

The speed-up, S, achieved through suppressing calculation of integrals that are multiplied by zero traction boundary conditions is summarised in Figure 8.5. It should be noted that the integrals that multiply zero displacement boundary conditions are not suppressed, as these values are required in the rigid body motion summation which is used to calculate the diagonals of $[H]$. Good speed-up is achieved, especially where n_{ui}/n is small.

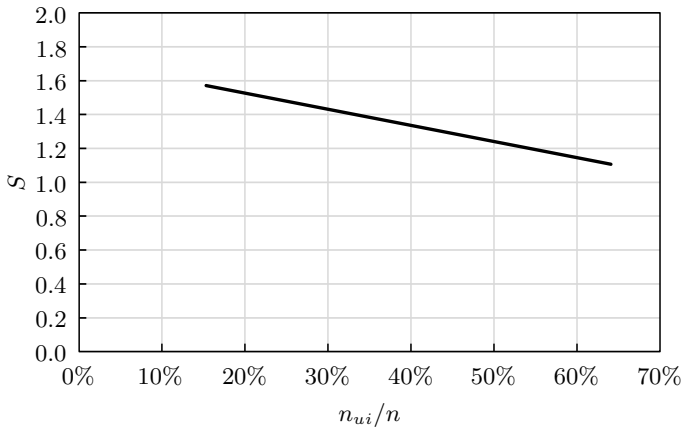


Figure 8.5: Speed up through suppression of integrals.

In Figure 8.5 it appears that if $n_{ui}/n > 70\%$, $S < 1$. Due to the algorithm's implementation this can never occur so $S \not< 1$.

8.5.4 Outer integration loop

For the majority of models, $S \approx 1.05$ if the outer loop of the integration algorithm is taken over the nodes instead of over the elements although, for small models ($n < 500$), S can be as much as 1.35. However, if the Bu and Davies integration scheme is used with the outer integration loop over the nodes it actually performs, on average, 20% faster than integrating primarily over the elements in conjunction with the same scheme. Node order integration also parallelises more efficiently than element order integration. An outer integration loop over the nodes has therefore been adopted for use in the accelerated algorithm.

8.5.5 Precision

Using single precision variables is on average 20% faster than using double precision variables. However, the size of the L_2-norm error in $\{x\}$, ε_x, associated with the reduction in precision increases exponentially with the number of nodes in the model, as shown in Figure 8.6. For the sizes of model encountered in this work single precision accuracy is sufficient. However, due to the exponential growth, double precision should be applied to larger problems. It is not easy to define a specific size of problem when double precision must be adopted as this is dependent on the desired accuracy, $\bar{\varepsilon}_x$. From Figure 8.6 the following formula can be constructed:

$$n \leq \frac{\ln(\bar{\varepsilon}_x) + 11.33}{0.0007} \tag{8.4}$$

where $\bar{\varepsilon}_x$ is the prescribed mean accuracy and n is the maximum number of nodes in the model. If this relationship is satisfied then $\varepsilon_x \leq \bar{\varepsilon}_x$.

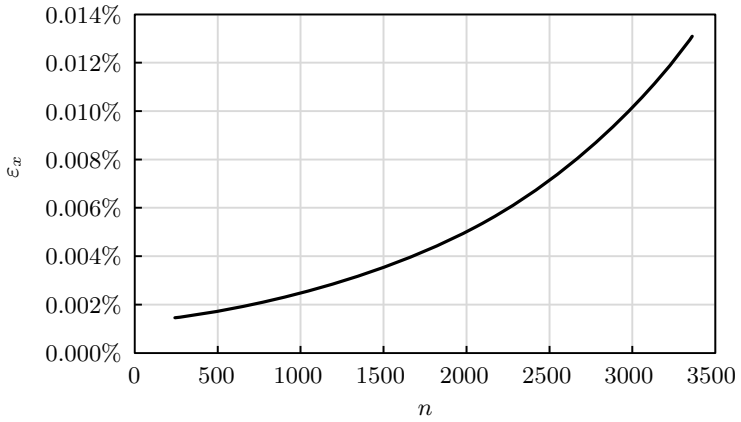


Figure 8.6: Increase in error using single precision.

It should be noted that using all single precision values in an iterative solver routine can reduce the convergence rate, although the solution accuracy is maintained. If double precision variables are retained when summing values during the matrix multiplication routine this reduction in speed can be overcome.

8.5.6 Architecture

Figure 8.7 shows the speed-up, S, achieved when using 64-bit processing instead of 32-bit processing. If $n > 900$, $S = 1.3$. However, for problems where $n < 450$, 32-bit processing is faster. Using the algorithm proposed in this work, models where $n > 1800$ must be computed using 64-bit processing as they require more than the maximum amount of memory that can be addressed using 32-bit processing. As $n > 450$ in the majority of problems, a 64-bit system architecture should be used wherever possible although some computer systems may only support 32-bits.

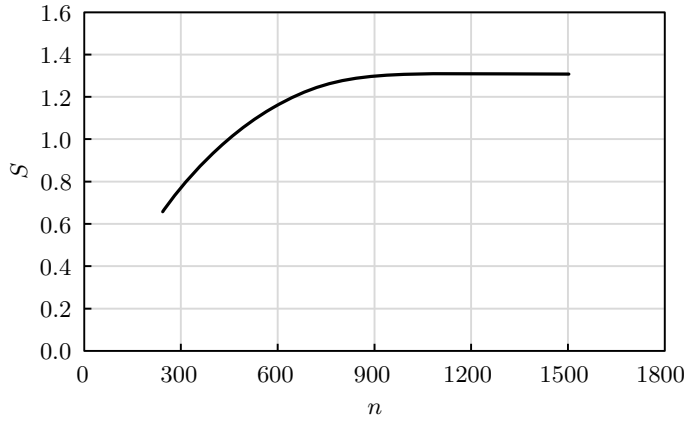


Figure 8.7: Speed up using 64-bit computing.

8.5.7 Summary

Previously in this thesis the acceleration schemes have been independently applied to the re-integration algorithm. This section combines them into a single algorithm.

Figures 8.8 and 8.9 summarise the results of applying each scheme independently. The vertical bars show the range of the data, the boxes the upper and lower quartiles and the bars the mean value of the speed-up, S, or the error, ε_x. The reader is reminded that ε_x is the increase in error resulting from application of the acceleration schemes and not the absolute error. Figures 8.8 and 8.9 have been constructed using data gathered from a range of different models with different numbers of DOF. It should be noted that all significant relationships relating to changes in S and ε_x have already been portrayed in this work, the remaining speed gains or changes in error related to the different acceleration schemes are approximately constant across all models and are within the ranges shown in Figures 8.8 and 8.9. It should be noted that Figure 8.9 has been cropped at 0.14% to show the results clearly. The peak values not shown in Figure 8.9 are between 3.0% and 4.5%.

In Figure 8.8 the field labelled 'Re-integration' shows S when applying the partial re-integration scheme as opposed to carrying out a full re-integration over the entire model with $\hat{m}$ fixed at 13 or 16 for triangular and quadrilateral elements respectively. The speed gain for the rest of the fields is the additional acceleration that can be achieved in addition to using this scheme. The field labelled 'All' summarises S when applying all of the selected improvements (suppressed integrals; 64-bit architecture; node order integration; single precision; Bu and Davies RISP integration scheme; updating $\hat{m}$ with each re-analysis).

Figure 8.10 shows t_u for a range of geometries for the re-integration algorithm and acceleration techniques proposed in this work and the scheme used by Foster *et al.* in [128] which uses the same number of Gauss points for all integrations but does limit re-integration to updated elements only. The proposed algorithm is significantly faster, providing a typical speed-up of $S = 5$.

The percentage of time saved in the re-integration, t_{ui}/t_{u0}, is plotted in Figure 8.11 against the percentage of updated nodes, n_{ui}/n. This shows that the proposed scheme improves on the speed-up achieved by Foster *et al.* [128]. The effect of overheads that resulted in $t_{ui} > t_{u0}$ when $n_{ui}/n > 0.63$ in the Foster *et al.* [128] scheme has also been reduced. However the overheads still exist, which will result in $t_{ui} > t_{u0}$ as $n_{ui}/n \to 1$. This is in agreement with the findings of Trevelyan *et al.* [16], given in Figure 1.1 where the extra spread in re-analysis time is due to variability in the solve time, t_{si}.

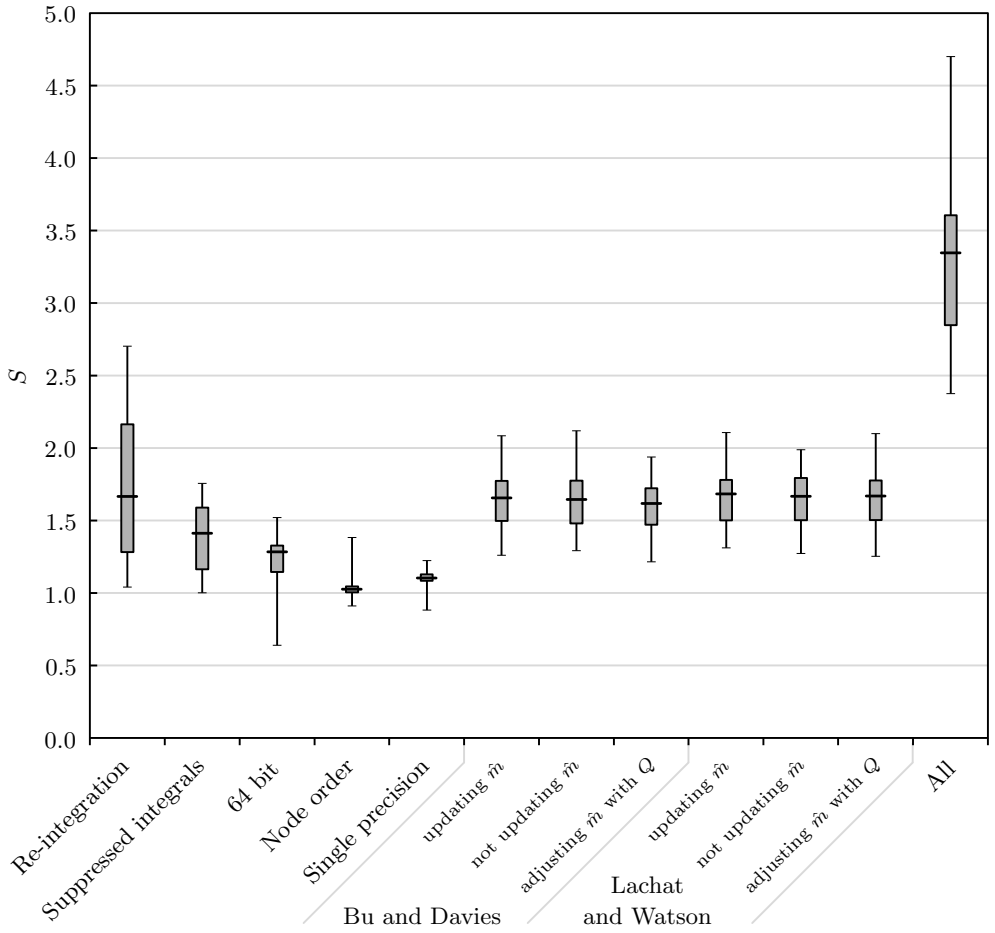


Figure 8.8: Summary of speed-up.

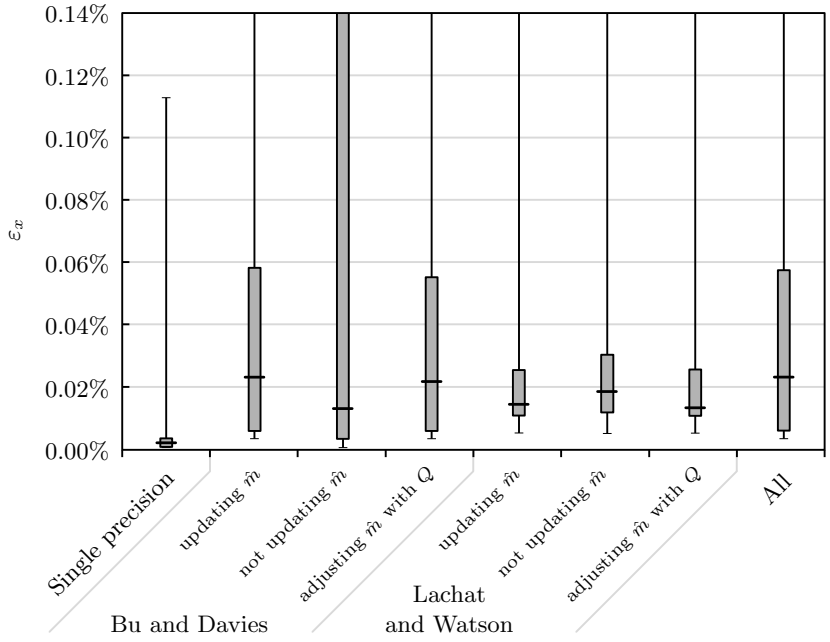


Figure 8.9: Errors associated with acceleration schemes.

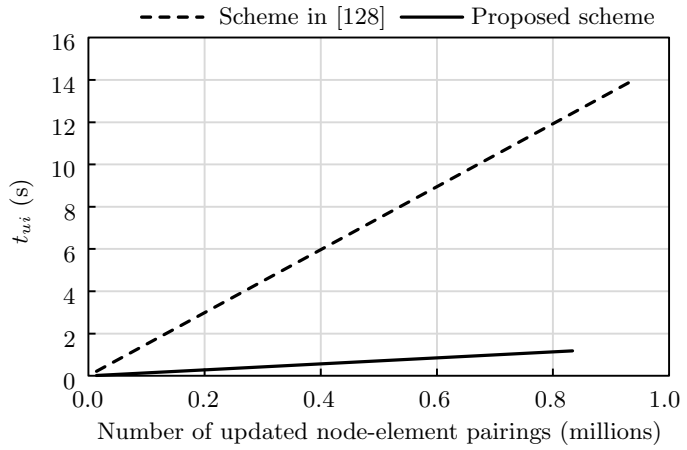


Figure 8.10: Absolute update time.

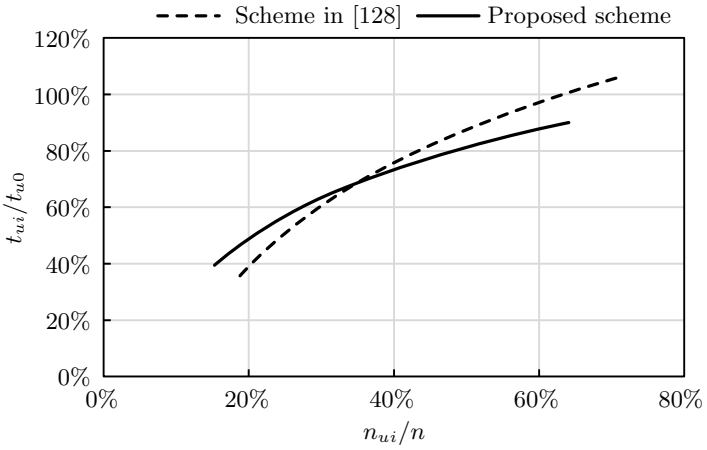


Figure 8.11: Normalised update time.

8.5.8 Parallelisation

The average speed-up, S, achieved by carrying out the integration on multiple Intel Xeon X5570 2.9GHz processors on a single shared memory machine is shown in Figure 8.12. For the re-integration, S is 15% smaller than for the initial integration. This is due to the overheads associated with parallelisation having a greater impact on the faster re-integration algorithm. Both schemes show good scalability which is independent of the percentage of the system that is updated.

8.6 Adaptive cross approximation

8.6.1 Introduction

Adaptive cross approximation (ACA) was developed for application in compressing the matrix, $[A]$, in a linear system of equations. This also enables acceleration of iterative solvers used with the system, as addressed in Section 5.4.5. ACA has traditionally only been applied to the $[A]$ matrix, and this is what is considered in this thesis. However, due to the mechanics behind the acceleration schemes already addressed in this chapter, it may now be preferable to apply ACA to the $[H]$ and $[G]$ matrices. This would have the effect of accelerating the integration and the assembly of $[A]$ as only the integrals required to construct the approximation would need to be calculated but could not be used to accelerate an iterative solver. This approach is not considered in this work but should be considered for further research.

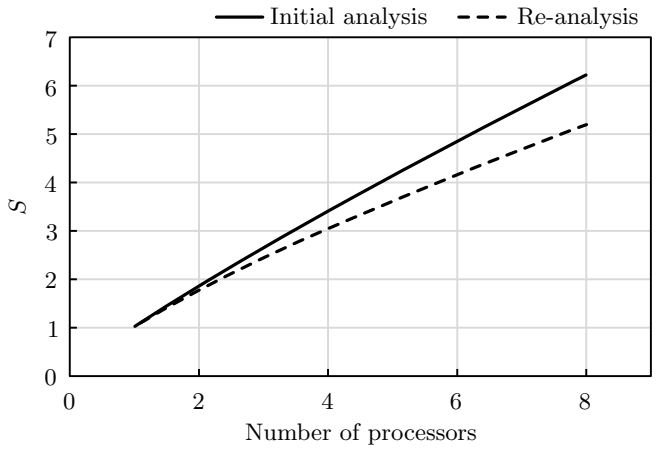


Figure 8.12: Integration speed gain for multiple processors.

8.6.2 Initial partitioning

The partitioning scheme presented in 4.3.3 is universal; it could be applied to any model. For this work we know more about the geometry of the model and are also able to predict which parts of the model the user is most likely to update. The matrix, $[A]$, can therefore be initially subdivided based on the geometry. The first level, or rank, of partitioning divides the nodes based on the patches on which they lie. Each patch may be subdivided further, this is done automatically, using geometric recognition to identify areas that are likely to be updated by the user. The nodes are then placed into appropriate partitions. An example of this approach is given in Figure 8.13 where it is predicted that the radius of the fillet is likely to be changed. If the fillet radius is changed, areas in white are unlikely to be updated, light grey will be partially updated and dark grey will be fully updated. Grouping these areas in this way helps to reduce the number of blocks that require updating as the geometry is changed.

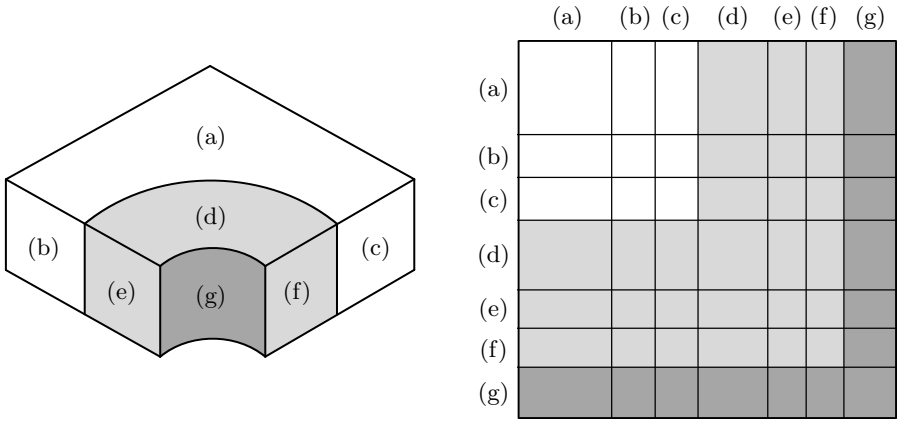


Figure 8.13: Initial partitioning of the model.

Once the algorithm has completed initial sub-division of the matrix, each block is, where necessary, subdivided further using the approach given in Section 4.3.3. The rank, R of a block is determined by the number of times it has been subdivided. A maximum rank, $R_{\max}$, is set to prevent an excessive number of small blocks being produced. The smallest blocks are always produced along the diagonal they will never be admissible. A minimum block size, $n'_{\min}$, is also set as blocks of only a few rows and columns are not worth approximating. To ensure that the change between consecutive integrals in the block is as smooth as possible, the nodes within each partition are renumbered traversing along the longest dimension of the

partition. The partitioning process adopted in this work is summarised in the flowchart given in Figure 8.14.

In addition to following the flowchart a few additional rules must be obeyed. Along lines, where nodes are shared by two patches, the nodes are partitioned with those on the patch to the left of the line. Along arcs, the nodes are always grouped with those on the curved patch, whichever side of the line the patch is on. This is because it is expected that the smoothest stress distribution, when comparing the two patches, will be around the curved surface. As the model partitioning will always produce a symmetric matrix block structure, the admissibility checks only need to be carried out once for each pair of blocks. Blocks located directly over the diagonal will never be admissible.

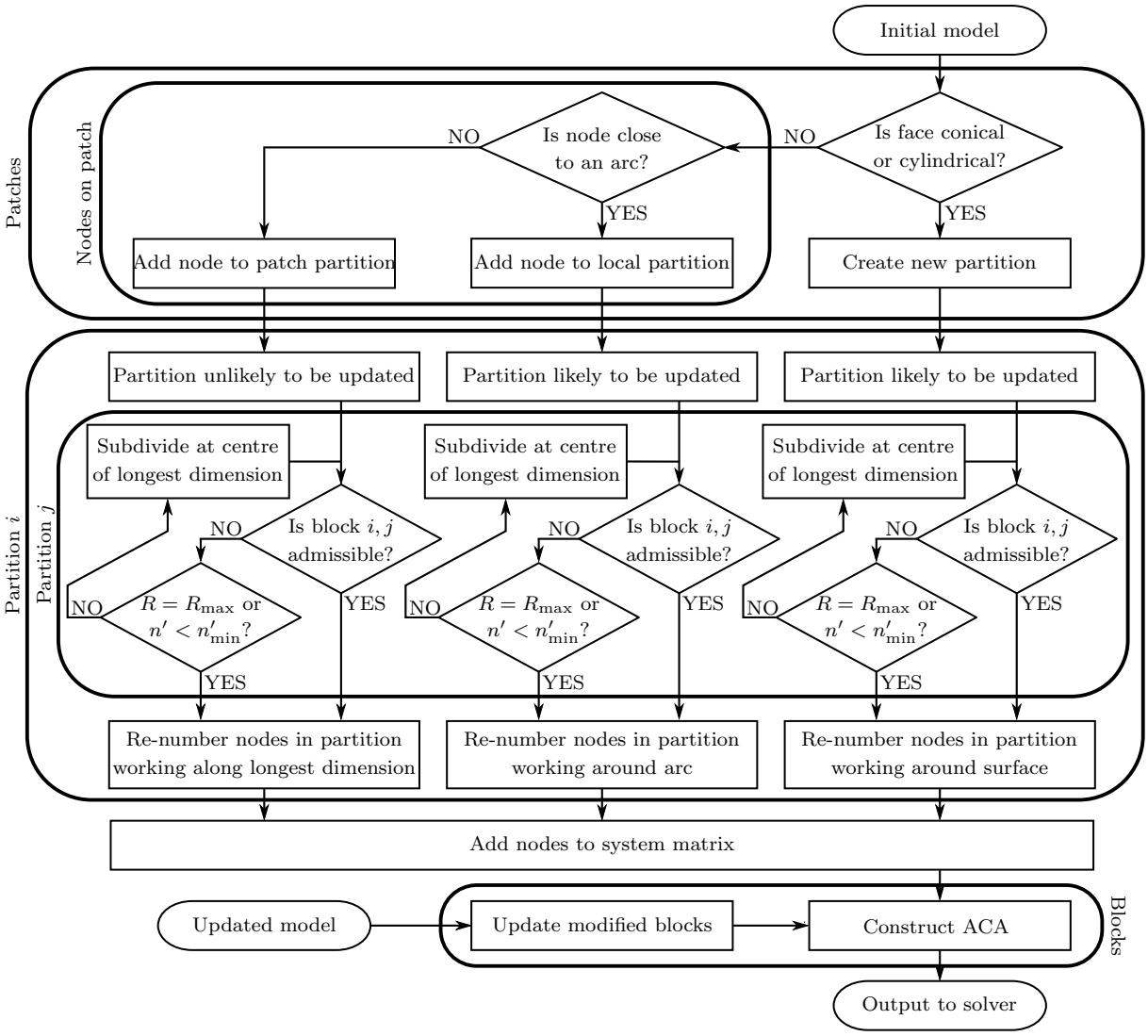


Figure 8.14: Block partitioning flowchart.

8.6.3 Construction

For the models assessed in this section, the factors that limit the size and number of blocks are set as follows: $R_{\max} = 10$; $n'_{\min} = 50$. Each block is approximated using Algorithm 4.1 with a tolerance, $tol = 0.001$.

Both the full and partial forms of ACA have been assessed here. The full ACA assumes that the

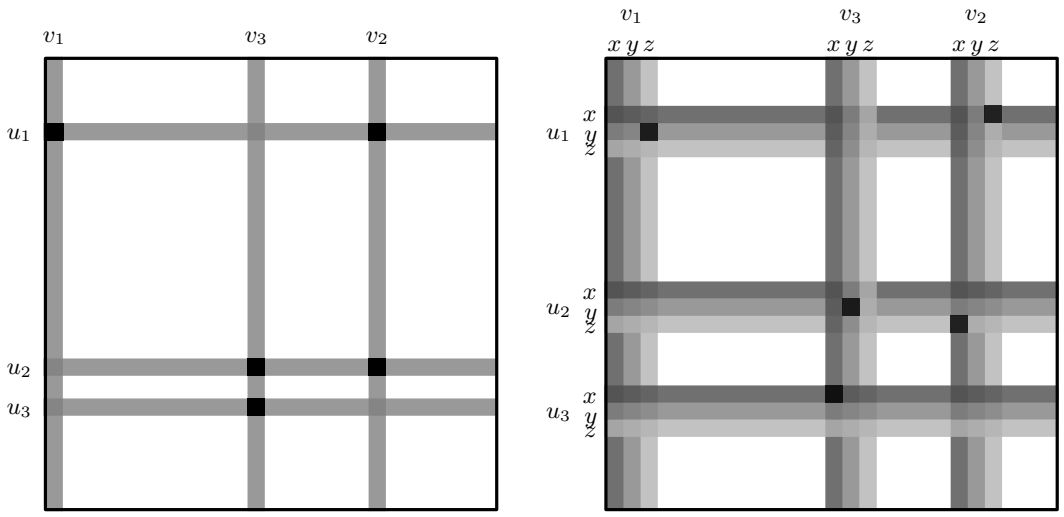
matrix, $[A]$ is already known and uses this to compute the error, ε_F, used to assess convergence of the approximation as described in equation (4.29) reproduced here:

$$\varepsilon_F = \frac{\|[S_k] - [A']\|_F}{\|[A']\|_F} \tag{8.5}$$

where $[A']$ is an admissible block and $[S_k]$ is the current approximation, computed from k rows and columns of $[A']$. The partial ACA assumes $[A]$ is unknown and checks against the previous approximation to establish if the approximation has converged:

$$\varepsilon_{Fp} = \frac{\|[S_k] - [S_{k-1}]\|_F}{\|[S_k]\|_F} \tag{8.6}$$

Two different schemes have been applied to construct the approximation. The first uses the standard form of ACA which uses a single row, u_i, or column, v_i, at a time, finding the maximum value in v_i to select u_i and the maximum value in u_i to select v_{i+1}. This is shown in Figure 8.15(a). The partial ACA is initialised with vector v_1. The maximum term in v_1, shown in black, is then used to select u_1. The maximum term in u_1 is then used to select v_2 and the process continues. The second scheme has been modified to benefit from the small amount of extra computation required to calculate the integrals relating to all the DOF on each node which differ only in the shape function used in their computation. This produces groups of three rows and columns where the maximum value out of all three vectors is used to pick the next group of rows or columns. This is shown in Figure 8.15(b). The problem is initialised with the three vectors v_1 which are all associated with the same node. The maximum term from these vectors, which is in v_{1z}, is used to select the next set of vectors, u_1, which correspond to the three DOF associated with another node. These, in turn, are used to select v_2 and the process continues until ε_F converges.



(a) Standard approximation. (b) Modified approximation.

Figure 8.15: ACA vector generation scheme.

8.6.4 Results

The block structure of two models, a TWC with $n_N = 820$ nodes and a PWH with $n_N = 2256$, is shown in Figure 8.16. Both models are partitioned as described in Figure 8.14. The initial partitions, defined by patch and areas of the mesh likely to be updated, are shown by the bold lines. The rank is

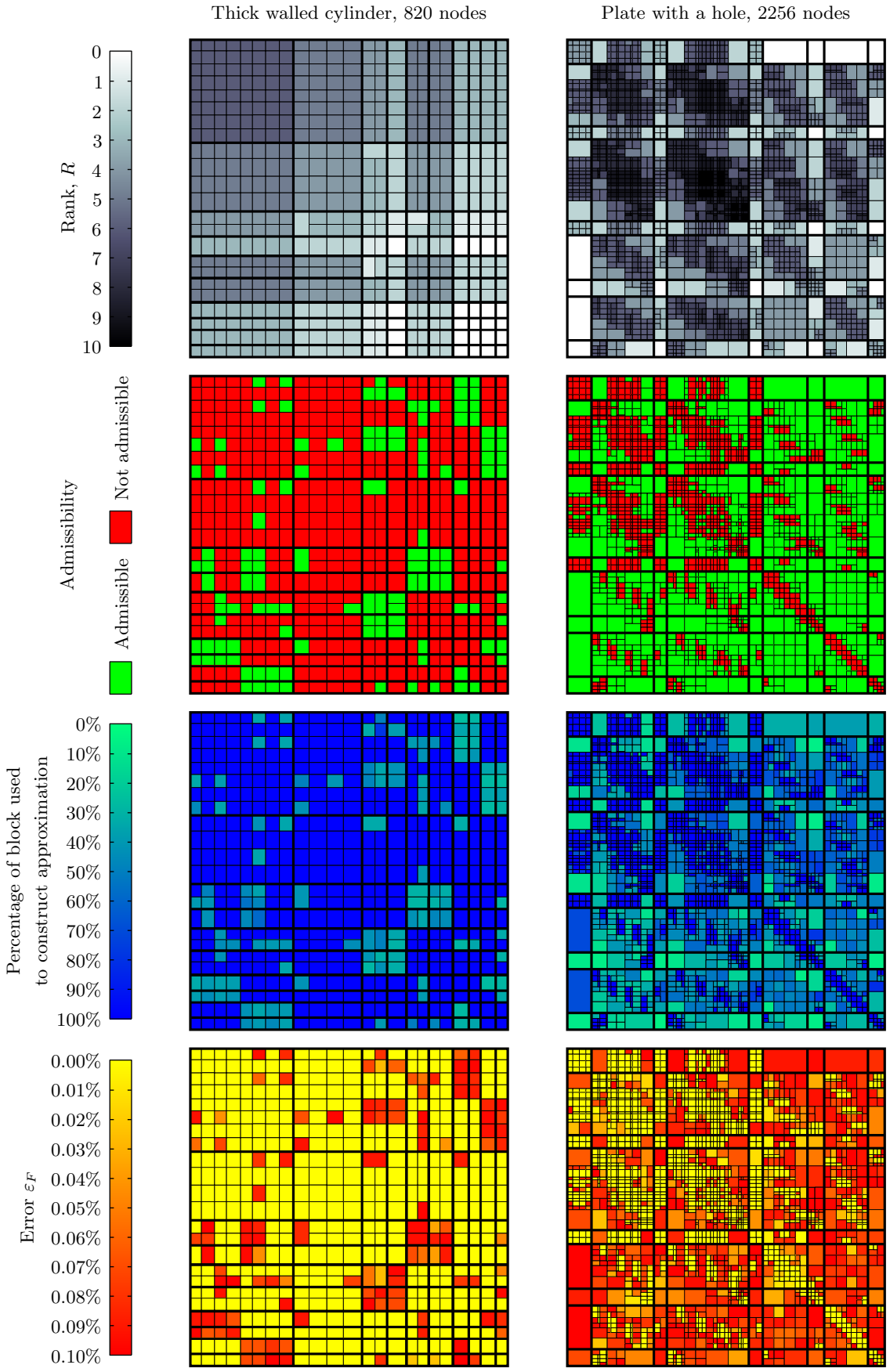


Figure 8.16: Block structure and admissibility.

limited to $R_{\max} = 10$. As can be seen in Figure 8.16, this does not affect smaller model as $n'_{\min}$ is the limiting factor. Smaller models also feature far fewer admissible blocks, as shown in Figure 8.17. All non-admissible blocks must be calculated in full and therefore show zero error in Figure 8.16.

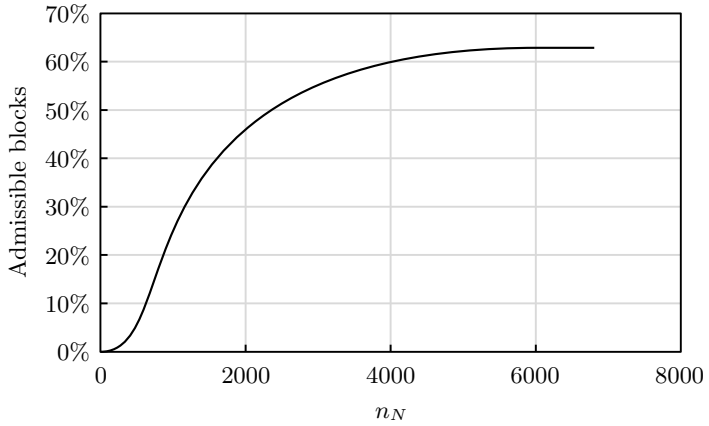


Figure 8.17: Percentage of blocks which are admissible for a TWC.

A similar number of vectors are required to approximate admissible blocks of any size. This leads to the potential for the approximation efficiency to improve with the model size, as shown in Figure 8.18, as larger problems feature larger blocks. If less than 50% of $[A]$ is generated, some acceleration can be achieved in the iterative solver.

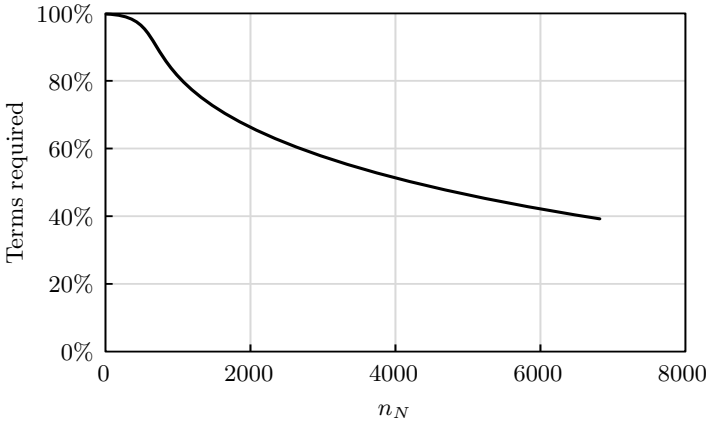


Figure 8.18: Percentage of the terms in $[A]$ required for ACA of a TWC.

The Frobenius error, ε_F, for each block is shown at the bottom of Figure 8.16 and for the entire matrix in 8.19. The error associated with non-admissible blocks is zero as these blocks are generated in full. Applying both forms of the full ACA produced almost identical errors. The partial ACA produces a much larger ε_F which can be reduced by applying the modified construction scheme.

Under some conditions the standard ACA scheme fails to converge. These conditions occur when a row or column of an admissible block contains values close to zero. The modified scheme does not experience these problems because it uses groups of three rows or columns at once.

The error in the stress, ε_σ, for the TWC model is calculated relative to the analytical solution using an L_2-norm. Figure 8.20 shows the value of ε_σ for different model sizes when using the full $[A]$ matrix or constructing $[A]$ from an ACA. The most accurate form of ACA is to use the full approximation with the modified approximation scheme described in Section 8.6.3. However, this requires generation of the full $[A]$ matrix. To accelerate the integration by only generating the required parts of $[A]$, the

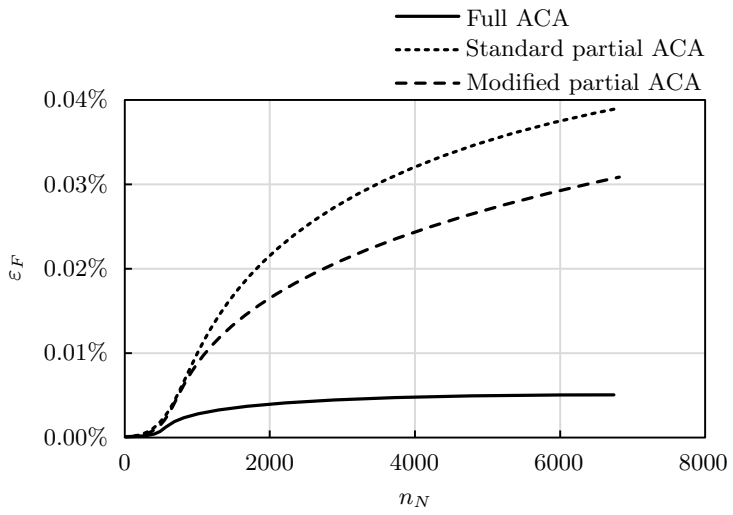


Figure 8.19: Frobenius error for ACA of a TWC.

modified ACA scheme must be applied. Using the full matrix always produces a smaller error than using an approximation.

Larger systems are approximated from a smaller percentage of the system, hence the increase in error with model size. The increase in ε_σ when $n < 1200$ is due to the mesh not being sufficiently refined to accurately capture the stresses. However, once $n > 1200$, ε_σ increases with n when using the complete $[A]$ matrix. This is the effect of applying the acceleration schemes discussed in Section 8.2.

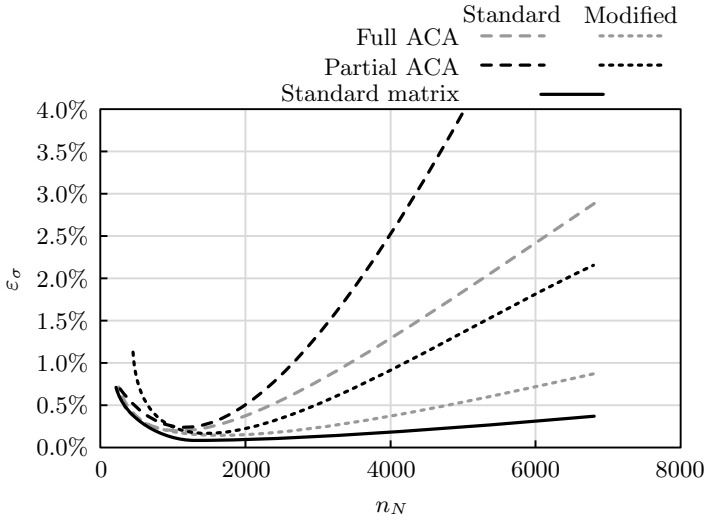


Figure 8.20: L_2-norm stress error due to ACA of a TWC.

The time required to construct the approximation is directly proportional to the number of admissible blocks and is much the same for all the approximation schemes discussed here. For the problem sizes assessed in this work it takes around 1.2 times longer to construct the approximation than it does to integrate the entire system. This does not take into account the time required to integrate the terms required for the approximation. ACA should therefore not be applied to models of the size encountered in this work. However, due to the increased efficiency in approximating larger matrices, ACA becomes more suitable as the problem size increases. However, further analysis of larger systems is required to establish the cut off point.

8.6.5 Adaptive cross approximation and re-integration

The most costly part of ACA is calculation of the error as this requires reconstruction of the current admissible block and calculation of the Frobenius norm which together require $3n'm's$ operations, where n' and m' are the number of rows and columns in the block and s is the number of pairs of vectors used to create the approximation. However, as the changes to the model will maintain the general geometric form and the element connectivity, a scheme is proposed whereby the same rows and columns as were generated during the initial analysis are re-used with the updated integration data to construct a new approximation. This removes the requirement to determine which vectors to use and the need to compute the error.

To assess this technique, a TWC meshed with 1426 nodes has been used as this is close to the optimum value for minimising ε_σ, as given in Figure 8.20. The TWC has an outer radius of 0.5m and an inner radius of 0.3m which is reduced to deform the model. An internal pressure of 1MPa is applied. The times to construct a new ACA and to construct the 're-cycled' ACA which uses the original rows and columns which now contain the modified data are shown in Figure 8.21. The time required to calculate the 're-cycled' approximation (without the re-integration time) is, on average, 30% of that required to construct an entirely new approximation. However, due to the time required to carry out the re-integration, the 're-cycled' scheme is slower than carrying out a full re-integration if more than 70% of the system is used in the approximation as is the case in Figure 8.21 which uses 72% of the system.

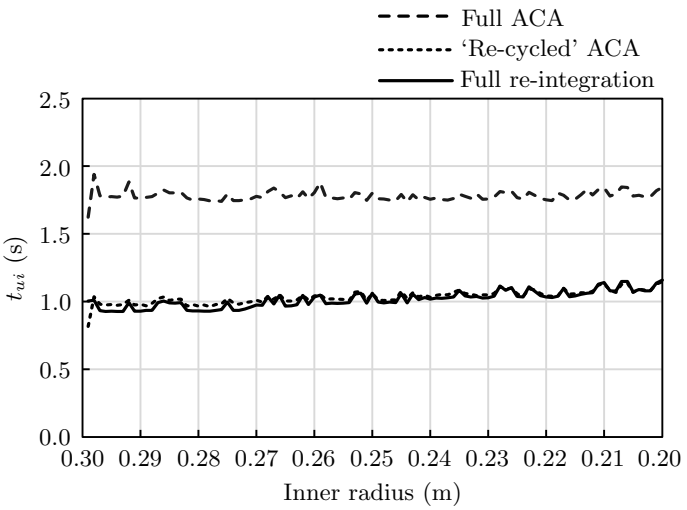


Figure 8.21: Time to re-construct ACA during re-analysis of a TWC.

In Figure 8.21, t_{ui} for the full ACA is shown to decrease with the radius. This is because the node groups defining the admissible blocks are moving further apart, thereby making the blocks smoother. If the inner radius were increased, t_{ui} would increase slowly for the full ACA.

As shown in Figure 8.22, generating a new approximation consistently maintains small ε_σ comparable in magnitude to a full re-integration. If n is increased, these will diverge as shown in Figure 8.20. The 're-cycled' approximation degrades as the model geometry increasingly moves away from the initial geometry. Erratic spikes in the error, where the approximation breaks down, can also be observed. These result in erroneous results at some nodes in the model, as shown in Figure 8.23 which shows the final update where the inner radius is 0.2m, whilst the majority of the stresses are correct.

These erratic errors in the stress results produced by the author's current implementation of the technique mean that it is not robust. However, the potential for speed gain means that the technique

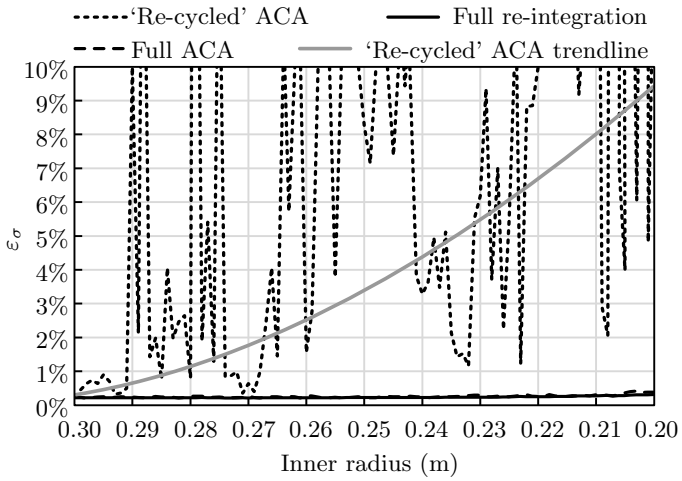


Figure 8.22: L_2-norm stress error during re-analysis of a TWC.

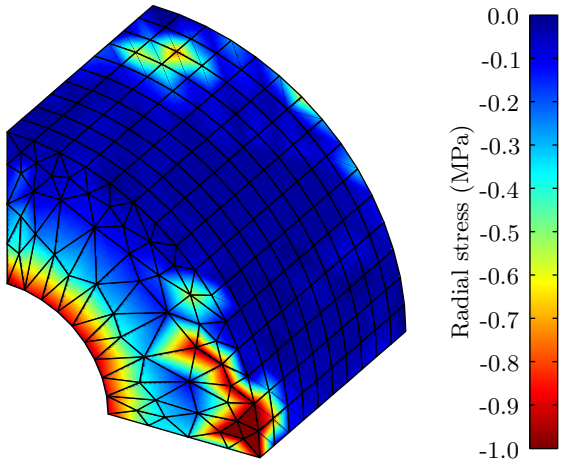


Figure 8.23: Erroneous stress contours caused by a poor approximation.

merits further research. It is possible that if a new ACA is calculated every few iterations then some of the increase in error could be overcome.

Chapter 9

Acceleration of the solution phase

"Nothing has such power to broaden the mind as the ability to investigate systematically and truly all that comes under thy observation in life."

Marcus Aurelius

9.1 Introduction

Linear solvers aim to solve the linear equation:

$$[A]\{x\} = \{b\} \tag{9.1}$$

There are several approaches to this, three of which are considered here. Direct solvers give an exact solution for the system but take a long time to execute. Iterative solvers make an initial 'guess' at the solution which they then iteratively improve upon. They are significantly faster than using a direct solver. Reduction techniques aim to approximate the problem as a basis of vectors. This reduces the size of the problem to one that can be solved in a fraction of the time although additional overheads are required to generate the basis.

The iterative solvers discussed in this chapter are usually applied to the finite element method (FEM). Here they have been applied to the boundary element method (BEM), which typically produces small, dense, non-symmetric matrices in contrast to the large, sparse systems found in the FEM.

The initial system of equations, $i = 0$, generated from the initial geometry, must be solved before any dynamic updating of geometry occurs. This is carried out using a full lower-upper (LU) decomposition. The LU decomposition is stored so that it can be used as a preconditioner in the re-analysis schemes.

9.2 Implementation

All of the solvers discussed in Chapter 5 have been implemented to assess their performance against each other. The pseudo-code for the author's implementations of these algorithms is presented in the appendix.

The iterative solver pseudo-code for the generalised minimal residual (GMRES), bi-conjugate gradient stabilised (BiCGSTAB) and transpose free quasi-minimal residual (TFQMR) schemes, presented in Algorithms A.1, A.2 and A.3, is based on the templates given in [149] with some modifications made by the author to improve efficiency. Leu's reduction, given in Algorithm A.4, eigenvector based proper orthogonal decomposition (E-POD) given in Algorithm A.5 and singular value decomposition based proper orthogonal decomposition (SVD-POD) given in Algorithm A.6, together with the proper orthogonal decomposition (POD) solution scheme given in Algorithm A.7, are based on algorithms found in [23], [25]

and [26] respectively. The nomenclature used in all the solvers is given in in Table A.1 and the solver specific nomenclature in Table A.2. Note that this may differ from the standard notation as some vectors have been reused to reduce the memory size and access requirements of the solvers.

For the type of systems encountered in this work it has been found that the GMRES method converges within 20 iterations in the worst case. The majority of solutions take 5-10 iterations, therefore no restarts are required. The amount of memory required by the solver, in addition to that required to store the system and preconditioner, increases with each iteration. For a typical 10 iteration solution, approximately an additional $12n$ double precision numbers need to be stored. This is small relative to the amount of memory occupied by the system and, for the sizes of system considered in the thesis, is far from prohibitive.

9.3 Test models

The test model considered to assess the linear solvers were the same as the models used to assess the re-integration of the system, as described in Section 8.4.

In addition to these models, a more detailed plate with hole (PWH) model has been used to assess how the linear solvers react as the model is deformed steadily further from the original geometry. This model was introduced in Section 7.4.

An error measure, ε_x, is used to compare the accuracy of each linear solver to a benchmark solution, $\hat{x}$:

$$\varepsilon_x = \frac{\|\{x\} - \{\hat{x}\}\|}{\|\{\hat{x}\}\|} \tag{9.2}$$

where $\|\cdot\|$ denotes the L_2-norm and $\{x\}$ the approximation to $\{\hat{x}\}$. The benchmark solution is calculated using a full Gauss elimination to solve the same set of equations.

The nomenclature used in the following sections will be as follows: the times, in seconds, to solve $[A_i]$ are denoted t_{si}. For the first iteration, $i = 0$, t_{s0} is the time to solve the system using a full LU decomposition. When $i > 0$, t_{si} is the solve time of the appropriate solver. The number of updated degrees of freedom (DOF) at the ith update is given by n_{ui}.

9.4 Comparison of the solvers

9.4.1 Overview

Two types of test were carried out on each model. The iterative algorithms were timed for a fixed accuracy, $\varepsilon_x < 0.005\%$, and ε_x was compared for a fixed solve time of 0.05 seconds. The timings do not include the time required to evaluate the new integrals. The decomposition methods are not iterative, therefore the accuracy cannot be prescribed in advance. These methods will also have a constant solve time for any given n.

Leu [23] only calculates the theoretical speed-up of his algorithm relative to a direct method based on the number of floating point operations. On implementation, the current author found that the greater number of memory accesses required by the algorithm and the order in which the data must be retrieved from memory had a detrimental effect on the speed of the algorithm. It could therefore not compete with the other methods for the type of problem considered here. Due to these considerations the Leu algorithm will not be discussed further in this work.

9.4.2 Fixed solver accuracy

The updated systems generated through geometric perturbation of a range of test models were solved using the iterative solvers and timed for a fixed accuracy, $\varepsilon_x < 0.005\%$. Figure 9.1 shows the relationship

between t_{si} and n, for the diagonally and LU preconditioned iterative solvers after the first geometric update has been applied to each test model ($i = 1$). This relationship is of the form:

$$t_{si} = cn_{it}n^2 \tag{9.3}$$

The constant c depends on the solver used. The value n_{it} is dependent on the solver and the condition of the preconditioned system and is the number of iterations carried out by the solver. It should be noted that Figure 9.1 shows a typical fit to aid visualisation of the general trend. There is some variation in the actual data. For the diagonally preconditioned systems n_{it} is approximately constant for the test problems in this study, as seen in Figure 9.2, which shows t_{si} for the plate with a hole model. For the LU preconditioned systems, the conditioning will initially degrade as the size of the perturbation from the original model geometry increases, causing n_{it} to increase. However once the geometry has changed significantly from the original, this degradation in the conditioning of the system will cease to substantially affect the solve time. This occurs at $d/D = 0.25$ in the case illustrated in Figure 9.2. However, the LU preconditioner continues to be a good preconditioner.

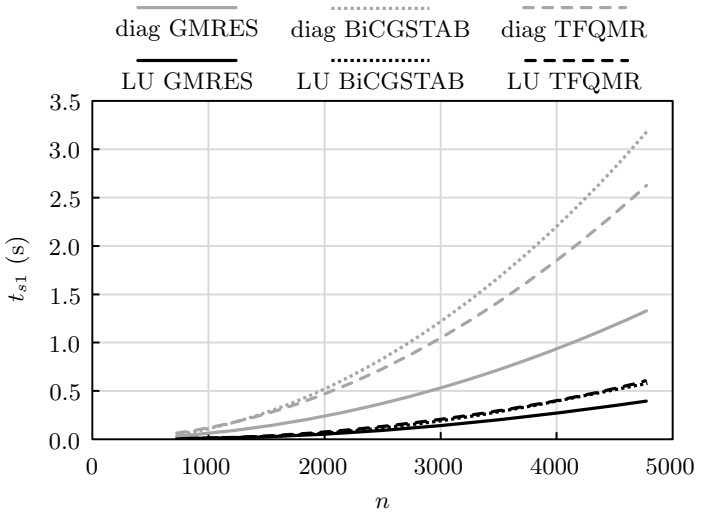


Figure 9.1: Comparison of iterative solve times for a fixed accuracy.

Where the time is available after the initial model is finalised and before re-analysis runs, a full LU decomposition should be computed since, when applied as a preconditioner, this can halve the re-analysis time for small geometric changes, as shown in Figure 9.2. However, if the model has small n, t_{si} will be more similar for the LU and diagonally preconditioned systems and, if a large modification is applied to a model with small n, the resulting system can be solved more quickly using diagonal preconditioning. This is due to the fact that in a small system a much higher percentage of the mesh will be regenerated after a geometric perturbation. The matrix, $[A_i]$, will therefore rapidly cease to bear much relation to $[A_0]$ and the LU and diagonal preconditioning will produce systems of comparable condition. This effect was only observed in the smallest ($n = 726$) of the 32 test cases assessed for this work and then only for the largest deformation. As the effect was negligible and $n > 726$ for the majority of models it can be safely assumed that LU preconditioning should be used to produce the smallest t_{si}.

If the total number of re-analysis updates, k, is known and the aim is to reduce the total run time:

$$t = \sum_{i=1}^{k} t_{ui} + t_{si} \tag{9.4}$$

then diagonal preconditioning should be used in preference to LU preconditioning if $k < n/40$, as this will provide the same accuracy for a reduced t. During dynamic updating of the model k will not be known and, as the aim is to minimise $t_{ui} + t_{si}$, LU preconditioning should be applied. However, the LU preconditioner must be recomputed sparingly (if at all), as it is expensive to compute, and only after a minimum of $n/40$ updates have been applied, to maintain the efficiency of the process. Alternatively a new LU preconditioner can be generated by a separate process in a new thread [17], thereby not detracting from the re-solution time.

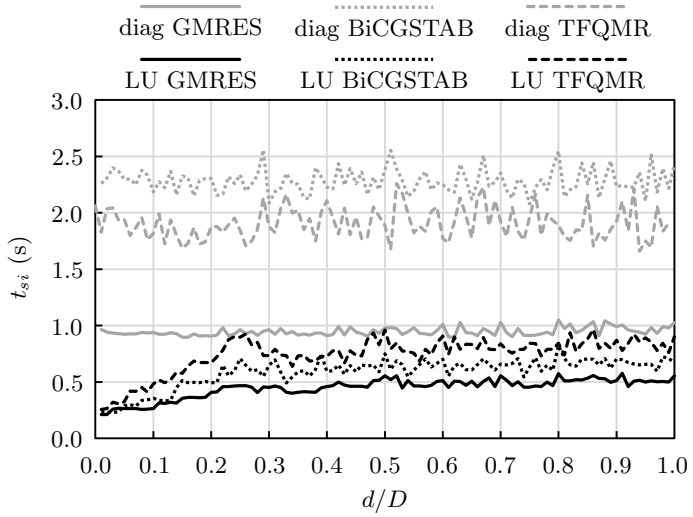


Figure 9.2: Resolve timings as a hole is moved along a thick plate ($n \approx 4300$).

Re-solve times and accuracies are model specific and different solvers may perform better under specific conditions but overall the GMRES approach most consistently provides most rapid convergence and has proved to be the most stable algorithm. Using these methods it appears that currently models of size $n < 2000$ are amenable for real-time update of results.

9.4.3 Fixed solve time

For rapid analysis, the solver will be required to re-solve the system to a sufficient accuracy within an acceptable time limit. To simulate this the iterative solvers have been allowed to run as many iterations as possible within 0.05 seconds. Reduction methods are not assessed here as the reduced problems always require a specific amount of overhead and are small enough to solve with a direct solver.

Figure 9.3 shows a diagrammatic representation of the error, ε_x, for a range of system sizes, n, based on numerical results. The contours depict the lower bound on ε_x for each solver. For LU preconditioned GMRES and BiCGSTAB solvers ε_x is consistently less than 0.1% (except in the occasional case). The error, ε_x, increases with n as larger systems require more time per iteration of the solver and therefore execute fewer iterations.

9.4.4 Solution of the system using proper orthogonal decomposition

For the first set of tests, the basis, $[\Psi]$, has been generated from the results of the first s analysis runs using both the E-POD and SVD-POD schemes. The re-solve time, t_{si}, is independent of $[\Psi]$ and i and is of the form:

$$t_{si} = cmn^2 \tag{9.5}$$

where c is a constant. The variation in ε_x between the two schemes is typically of the order 10^{-6} when using the same m.

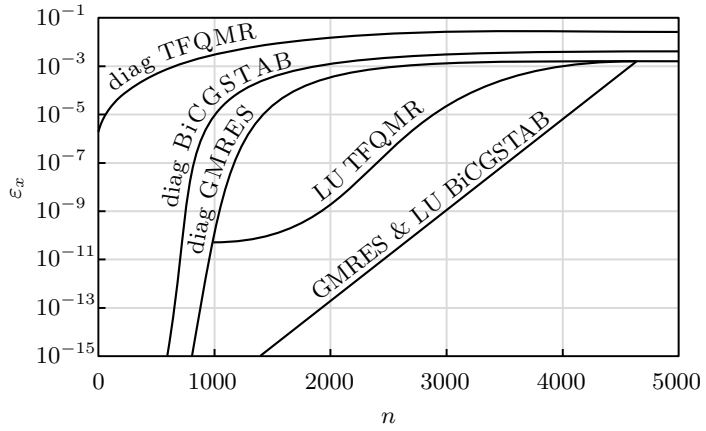


Figure 9.3: Comparison of iterative solve errors for a fixed solve time.

As shown in Figure 9.4, the time required to generate $[\Psi]$, t_B, is large when applying E-POD as the eigenvalues of an $n \times n$ system must be computed. As t_B increases cubically with n, this is prohibitive for anything but the smallest models for this analysis application. When applying SVD-POD the singular value decomposition (SVD) of an $n \times s$ system must be found; this leads to a considerably smaller t_B than E-POD. When computing the SVD-POD, t_B appears to bear little relationship to n for systems of the size encountered in this study ($n < 5000$) and is typically less than 0.1 seconds.

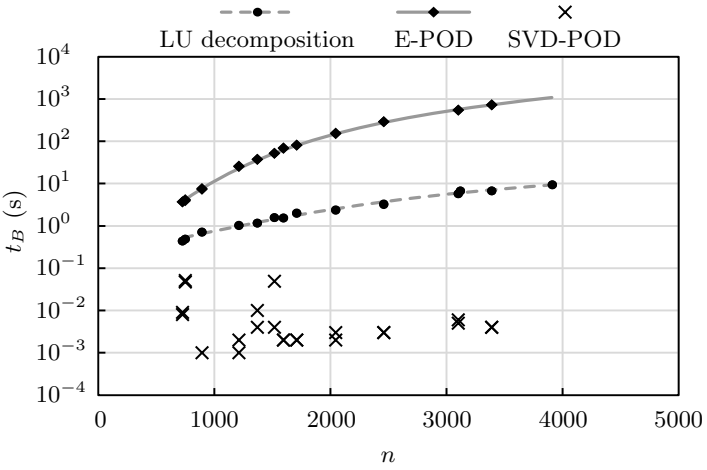


Figure 9.4: Time to generate $[\Psi]$ for different system sizes.

System size, n, has little effect on ε_x which is mainly influenced by the number of basis vectors, m, and increases rapidly with i. The benefit of increasing m is limited, for example, in Figure 9.5, if $m > 3$ no reduction in the error is observed once $d/D = 0.08$. Both Kerfriden *et al.* [26] and Ryckelynck *et al.* [25] propose enrichment schemes to regulate ε_x by adding vectors to $[\Psi]$ based on the latest analysis results. However, these enrichment schemes have not been implemented in the current work as any increase in t_{si} will render the method uncompetitive with the GMRES algorithm for the types of system encountered in this study.

Figure 9.6 has been produced from the test model shown in Figure 7.13 by moving the hole in the direction $x = y$. The error, ε_x, is compared for the SVD-POD with $m = 3$ to ε_x for the LU and diagonally preconditioned GMRES algorithms. The same t_{si} has been applied to the GMRES algorithms as was used to solve the reduced system. It can be seen that, for this model, once $d/D = 0.08$, the GMRES

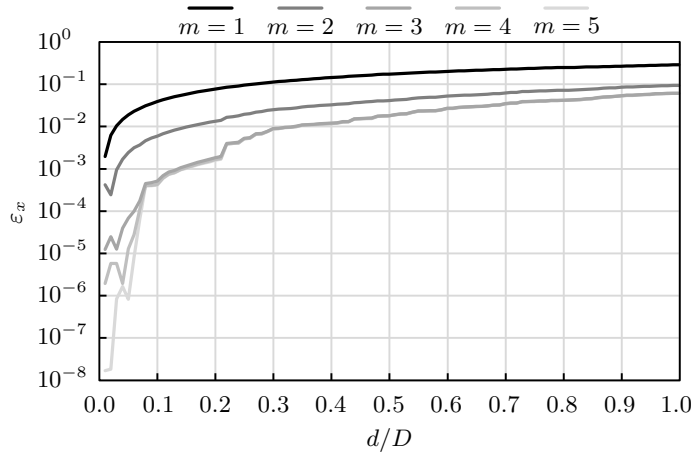


Figure 9.5: Change in solution error, ε_x, as a hole is moved diagonally across a thick plate.

methods provide a more accurate solution and ε_x does not degrade as rapidly. If a larger value of m were used these effects become even more pronounced.

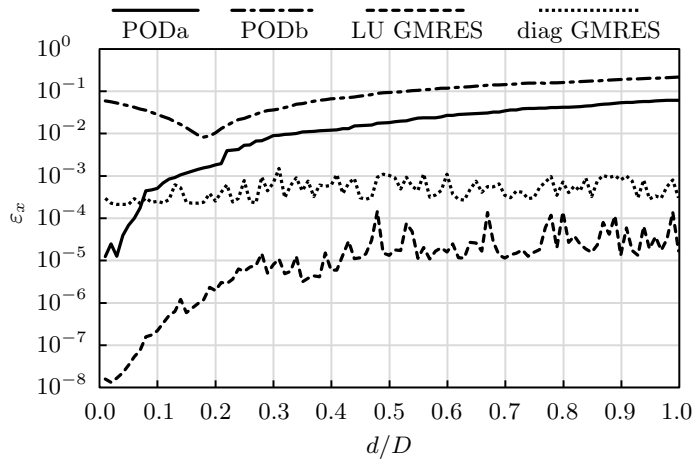


Figure 9.6: Solution error for a fixed solve time ($m = 3$).

In Figure 9.6, PODa denotes a set of runs using the first four analysis results to generate the basis, whereas PODb considers bases drawn from a set of representative solutions generated from the following models:

1. The initial model.
2. The hole was moved in the x-direction through distance $D/2$.
3. The hole was moved in the y-direction through distance $D/2$.
4. The plate was stretched in the x-direction through distance D.

PODb does not provide any improvement in ε_x over PODa. A poorly selected basis can also lead to an even greater ε_x.

9.4.5 Summary

Of the solvers assessed in this section, the LU preconditioned GMRES solver provides the fastest and most accurate solution. It has therefore been adopted as the primary solver for re-analysis ($i > 0$). On the initial analysis run ($i = 0$), a complete LU decomposition is used to solve the system. This generates

an accurate starting point but also produces the LU preconditioner which has been applied to accelerate the GMRES scheme.

If a one off analysis is required, without the need to later update the model geometry, a diagonally preconditioned GMRES solver is applied as this can be used directly with the initial analysis and does not require the costly generation of the LU preconditioner.

9.5 Parallelisation

9.5.1 Introduction

It has been shown that the GMRES algorithm is the most suited to the type of problems encountered in this work. For this reason, only the acceleration that can be achieved by the GMRES algorithm and the LU decomposition required to produce the LU preconditioner will be considered in parallel. It can be shown that parallelising the other iterative methods would not result in a greater performance increase than using the GMRES algorithm in parallel. If this were not the case using another method could lead to a faster solve when applied in parallel.

9.5.2 On multiple CPUs

Only parts of the solvers can be parallelised for the central processing unit (CPU). The overheads required to solve a parallel version of the forward and backward substitution routine, used to apply the preconditioner in the iterative schemes and to solve the LU decomposition in the direct solver routine, make it impractical to parallelise this part of the algorithm for the size of problem encountered in this work. As this is the major contributor to the time required by the LU preconditioned GMRES algorithm, any speed gain made by parallelising the algorithm will be small. The speed-up factor, S, achieved by parallelising the algorithms on a single shared memory machine is shown in Figure 9.7

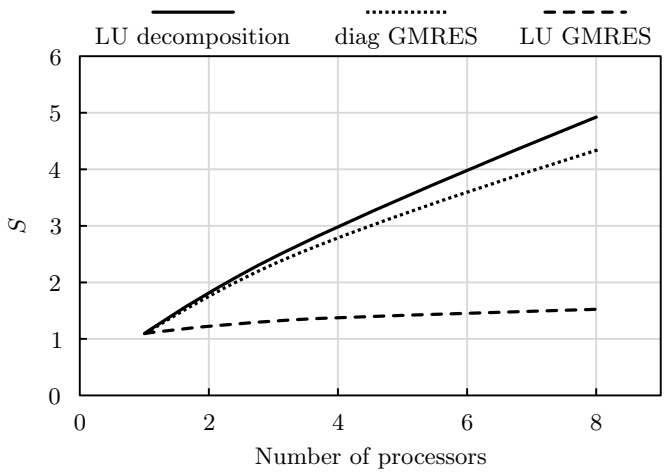


Figure 9.7: Speed up of LU decomposition and GMRES solvers on multiple CPUs.

Both the LU solver and the diagonally preconditioned GMRES algorithm achieve good speed-up and show good scalability when parallelised. It is possible that additional efficiency gains could be made through a more specialised implementation. However, this would require extra overheads which, when dealing with the small problems encountered in this work, are likely to negate any additional speed gain achieved.

The LU preconditioned GMRES algorithm parallelises far less efficiently due to the forward and backward substitution routine. This means that, on eight processors, the diagonally preconditioned

algorithm is competitive with the LU preconditioned algorithm. This is shown in Figure 9.8 which uses the plate with a hole model described in Section 9.3, with a best fit line through the results. For machines with fewer than eight processors, the LU preconditioned is always faster, as shown in Figure 9.1. For machines with eight processors the less predictable convergence time and the additional memory required by the diagonally preconditioned algorithm means that the LU preconditioned GMRES solver is still the best option. However, if more than eight processors are available the diagonally preconditioned solver will prove to be faster.

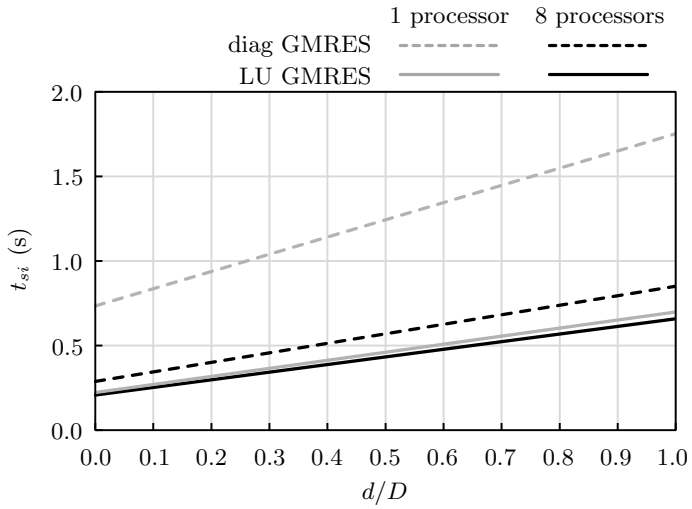


Figure 9.8: Best fit comparison of solve times on one or eight CPUs.

9.5.3 On GPUs

The performance of some of the linear solvers has been tested with a graphics processing unit (GPU) implementation. The full LU decomposition, used in the initial solve, and the LU preconditioned GMRES, BiCGSTAB and TFQMR algorithms, used for the re-solve, have been coded for GPUs. This has been achieved using Nvidia's *compute unified device architecture (CUDA)* and the *CUDA basic linear algebra subprograms (CUBLAS)* library [162]. Small calculations have been retained on the CPU along with the Givens rotations used in the GMRES algorithm as these operations are not as suited to GPU parallelisation.

On account of the small number of calculations involved in re-solving the system using POD it would be counterproductive to run this solver on the GPU.

Figure 9.9 shows the speed-up, S = CPU time/GPU time, for the re-analysis solvers with a range of system sizes, n. The CPU results have been generated using an Intel Xeon X5570 CPU with the author's solvers. The GPU results have been generated using an Nvidia Tesla C2070 GPU with the CUBLAS library. The relationship between S and n is approximately linear for systems of this size and it is found that it is only for systems of size $n > 5000$ that the GPU computation is beneficial. When generating and solving the LU decomposition used in the initial analysis, many more operations are applied to the same set of data and efficiency gains can be achieved for smaller systems of size $n > 2500$, as shown in Figure 9.10. The speed can be further improved if a blocked LU decomposition is used, where the block size, b, is chosen to fill the memory on the GPU. It is possible that using a similar blocking routine, such as an element-by-element form [168], with the iterative solvers could lead to a similar acceleration in these algorithms.

Some round off effects, as described by Smith and Margetts [169], were observed to affect the number

of iterations in the BiCGSTAB solver only. In the worst case these had a negligible effect on the error, ε_x, of the order 10^{-8}. No variation in the solution accuracy was observed between the CPU and GPU results for the GMRES and BiCGSTAB solvers.

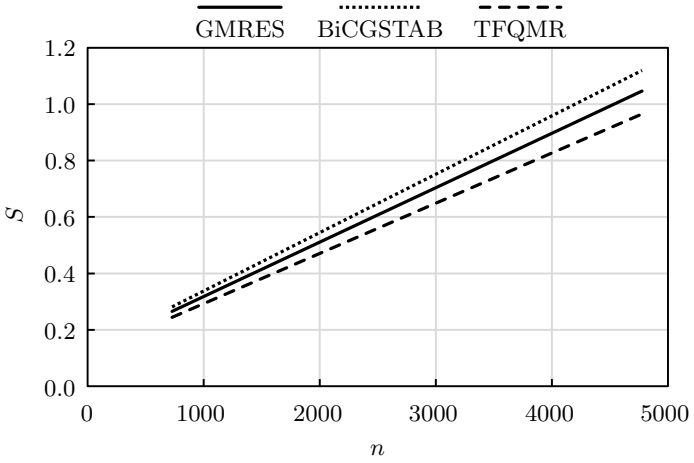


Figure 9.9: Comparison of CPU and GPU iterative solve times.

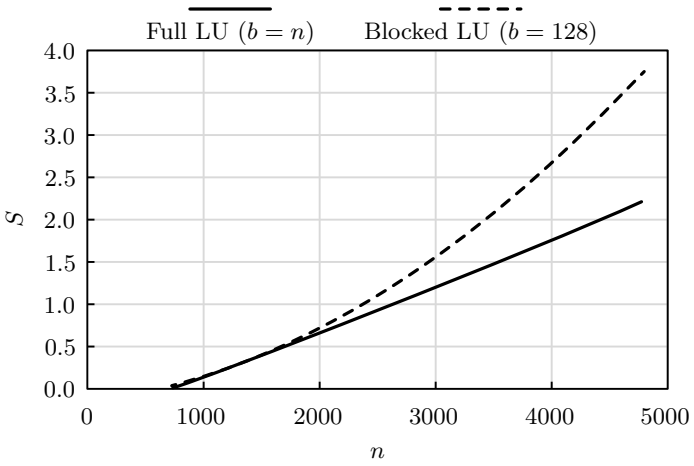


Figure 9.10: Comparison of GPU and GPU LU decomposition and solve times.

Chapter 10

Evaluation

"The study and knowledge of the universe would somehow be lame and defective were no practical results to follow."

Marcus Tullius Cicero

10.1 Introduction

So far in this thesis the re-meshing, re-integration and re-solution algorithms have been independently assessed to find the most efficient algorithms. This chapter aims to draw all these areas together to evaluate the software as a whole. Section 10.4 covers the integration of the algorithms developed in this work into the Concept Analyst 3D (CA3D) software package.

Throughout the development of the algorithms contained in this work, regular meetings have been held with industrial partners. These have been aimed at steering the development towards solving problems applicable in industry, informing the design of the graphical user interface (GUI) and suggesting typical model geometries that are likely to be encountered. Two case studies that have been developed as typical problems that might be encountered during the design of an aircraft are introduced in this chapter. The first is designed to show the benefits of the new re-meshing, re-integration and re-solution schemes whilst the second focusses more on the benefit to the analyst of being able to rapidly re-analyse a model to find an optimum solution to a problem.

10.2 Case study: Countersunk hole

10.2.1 The model

The countersunk hole model used in this case study is described in Figure 10.1. The angle of the countersink is $\pi/4$ radians. Only a quarter of the model has been meshed and analysed to reduce the complexity of the simulation. Roller boundary conditions are applied to the planes of symmetry which bisect the hole and to the base of the model. A nominal load of 10MPa is applied as indicated. The countersunk hole has been modelled using 540 elements with 3066 degrees of freedom (DOF).

For this model we consider the effect of changing the depth, d, of the countersink. Initially $d = 3$mm. The depth is then incrementally increased to establish how this affects the stress distribution. Data have been collected on all aspects of the algorithm; the results are summarised in this section.

10.2.2 Results

The principal stresses may be used to calculate the von Mises stress:

$$\sigma_{VM} = \sqrt{1/2((\sigma_1 - \sigma_2)^2 + (\sigma_1 - \sigma_3)^2 + (\sigma_2 - \sigma_3)^2)} \tag{10.1}$$

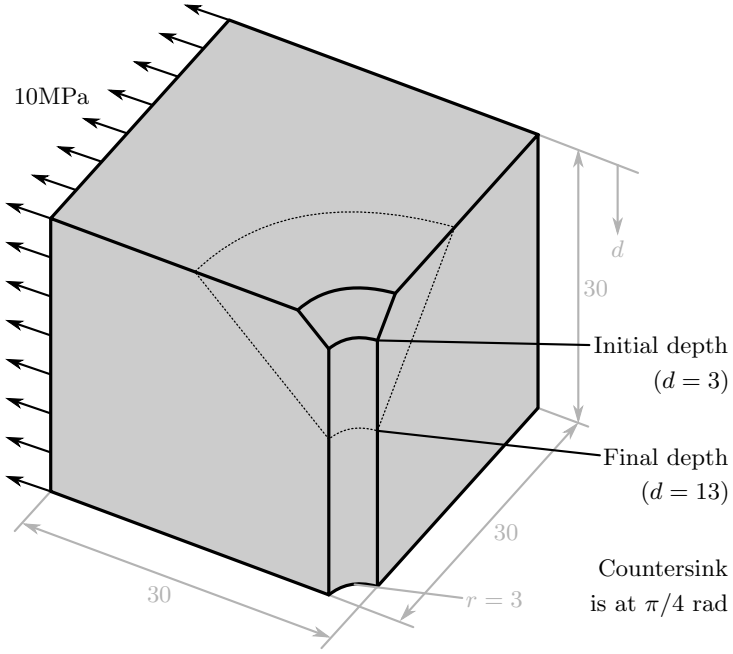
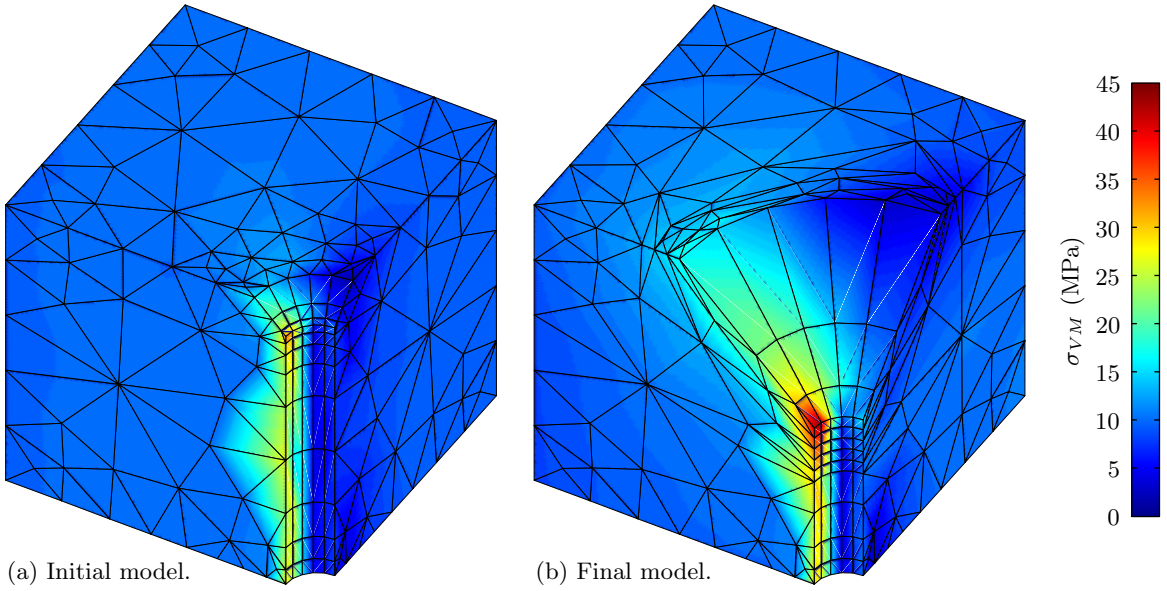


Figure 10.1: Case study: Countersunk hole. (Dimensions in mm)

across the surface of the countersunk hole model is shown in Figure 10.2. The stress concentration factor, K_t, is defined as the maximum von Mises stress divided by the applied stress. The highest stress concentration is apparent at the base of the countersink and increases almost linearly with d with the exception of $6.5 < d < 7.5$, as shown in Figure 10.3, and is typically within 5% of the value given in Pilkey and Pilkey [170], where the data has been generated through a series of finite element simulations. This suggests that, to minimise the stress, the countersink should be kept as shallow as possible, which is the intuitive result. The remainder of the stresses in the model remain almost constant as d is updated.



(a) Initial model. (b) Final model.

Figure 10.2: von Mises stress concentration over a countersunk hole.

It should be noted that the contour plot has been created by interpolating the values of stress at the nodes. The irregularities in these plots, especially apparent along the sides of the hole, are created by

this interpolation scheme. It is therefore imperative that the contour results are treated with caution, especially in areas of high stress gradient. The peak stress is within 5% of the value in Pilkey and Pilkey [170]. A more accurate mesh could be applied to reduce this error; however, the reader is reminded that a compromise has been made between the accuracy of the results during re-analysis and the speed of the analysis. The mesh given in this example illustrates the amount of distortion that can be permitted whilst maintaining 5% accuracy during real-time re-analysis. Once a satisfactory design has been established by the analyst, a more detailed analysis with a finer mesh can be carried out to ensure that the proposed model is viable.

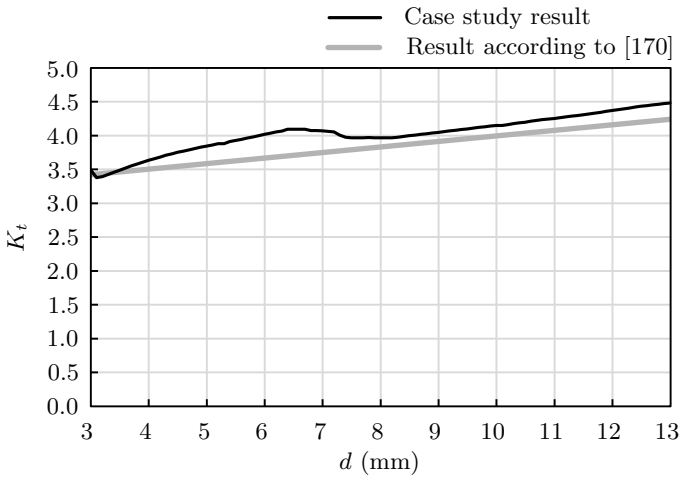
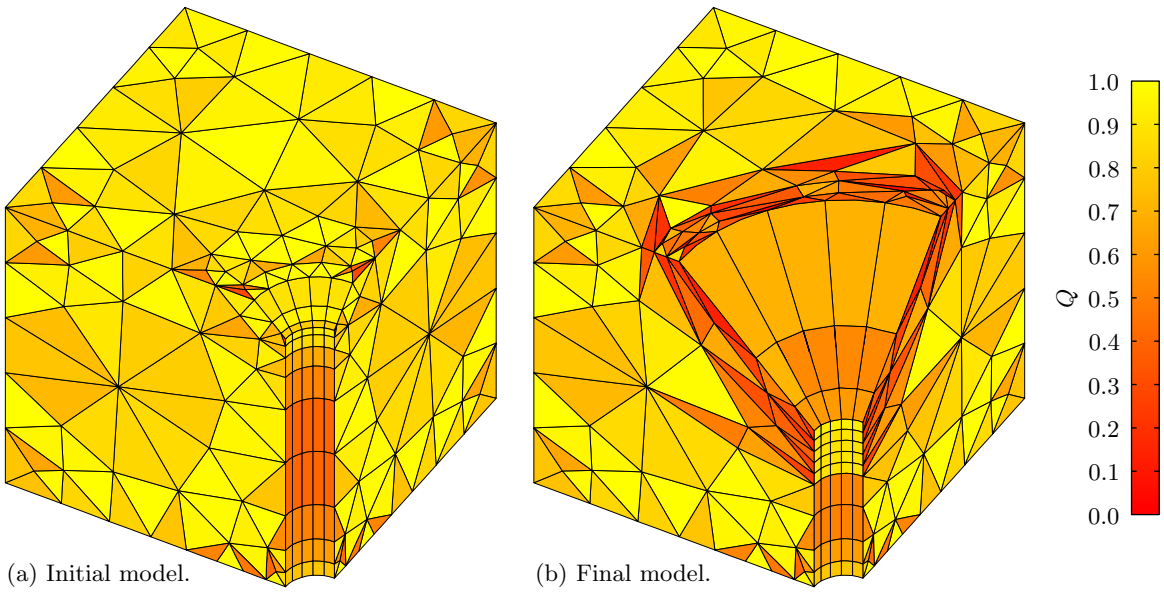


Figure 10.3: Stress concentration factor at the base of the countersink.

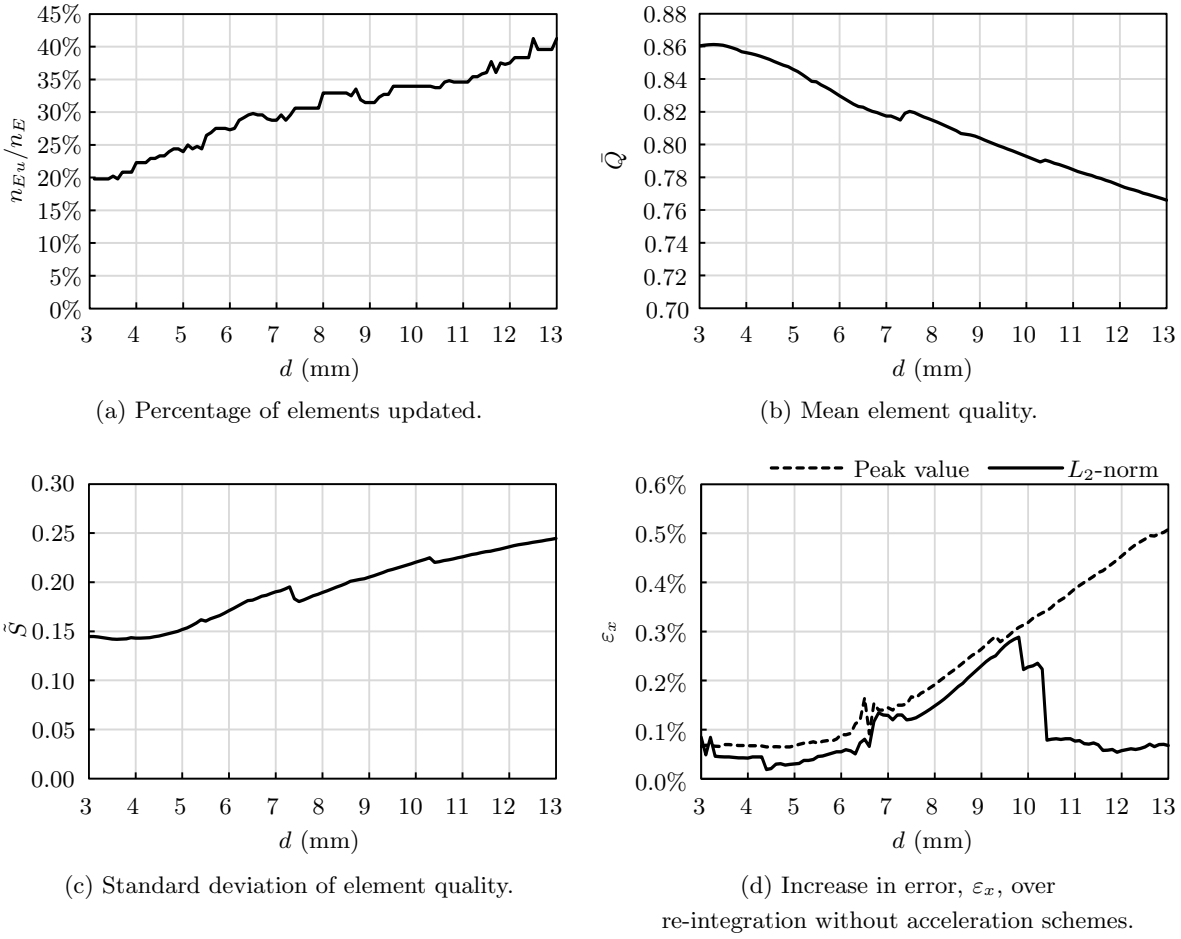
Figure 10.4 shows the distribution of element quality, Q, on the initial mesh ($d = 3$mm) and the final mesh ($d = 13$mm). The initial mesh is of high quality. The quality of the elements immediately adjacent to the countersink steadily degrades as the geometry is distorted but remains within acceptable limits. This is captured in Figures 10.5(b) and 10.5(c). However, local re-meshing would be required if d was increased much further.



(a) Initial model. (b) Final model.

Figure 10.4: Element quality over a countersunk hole.

The error in the solution vector, ε_x, calculated using equation (9.2), is shown in Figure 10.5(d). The increase is caused by the increased element distortion which results in elements of quality $Q < 0.25$ around some of the key geometrical features. At this level of distortion, the shape functions, which are equivalent to the basis vectors in which we are seeking a solution, cease to be rich enough to match the solution as closely as for high quality elements.



(a) Percentage of elements updated.

(b) Mean element quality.

(c) Standard deviation of element quality.

(d) Increase in error, ε_x, over re-integration without acceleration schemes.

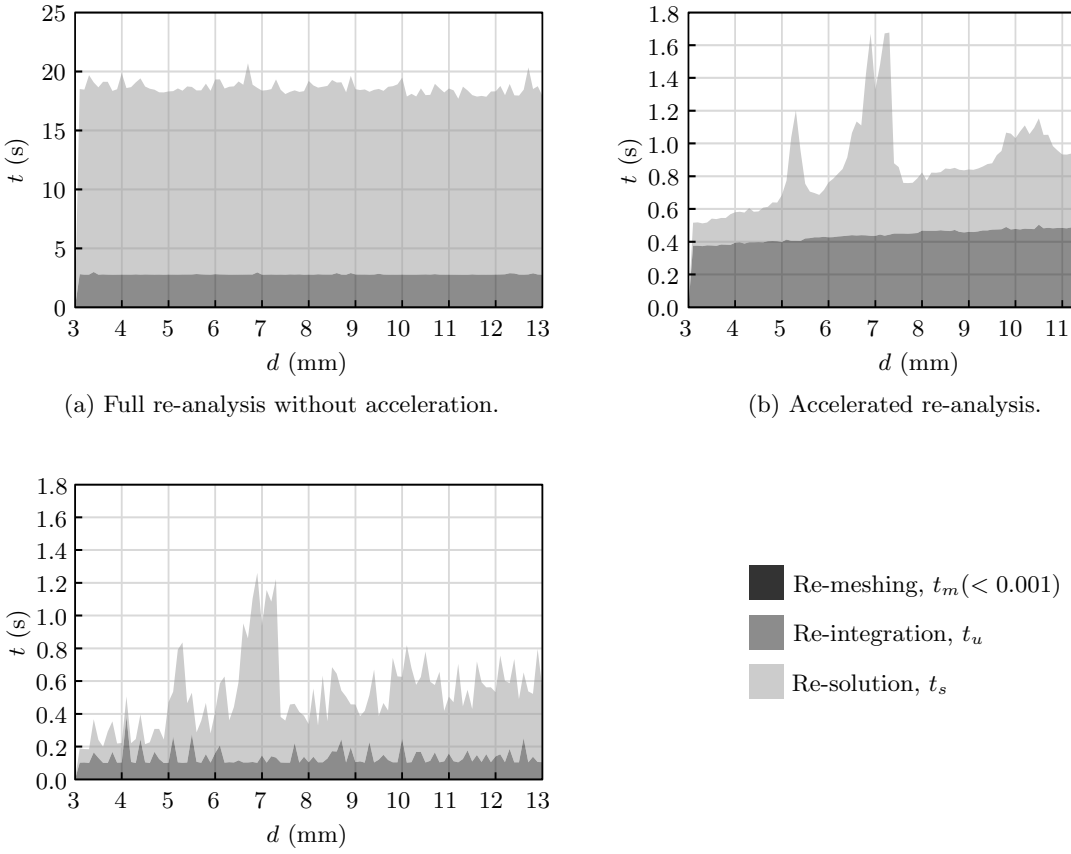
Figure 10.5: Mesh and re-meshing results for a countersunk hole.

10.2.3 Timings

The distribution of re-integration time, t, across re-meshing (t_m), re-integration (t_u) and re-solution (t_s) as the depth of the countersink is increased is shown in Figure 10.6. Figure 10.6(a) shows t using a full re-integration with a direct solver. The total analysis time, t, is clearly dominated by t_s. By applying the lower-upper (LU) preconditioned generalised minimal residual (GMRES) solver described in Chapter 9 instead of a Gauss elimination, this can be reduced by a factor of 45 for this model. If the integration acceleration schemes discussed in Chapter 8 are also applied, Figure 10.6(b) is produced. The timings t_u and t_s are now similar. However, the re-integration parallelises far more efficiently that the GMRES solver and in parallel the solution time once again begins to dominate as shown in Figure 10.6(c).

The time to re-mesh the model, t_m is less than 0.001 seconds and is constant for all three figures. It is included in Figure 10.6 but is too small to be apparent. The meshing and re-meshing schemes have not been parallelised as there would be little benefit to accelerating this algorithm further.

The speed-up, S, achieved by the new re-integration and re-solution schemes over a complete tradi-



(a) Full re-analysis without acceleration.

(b) Accelerated re-analysis.

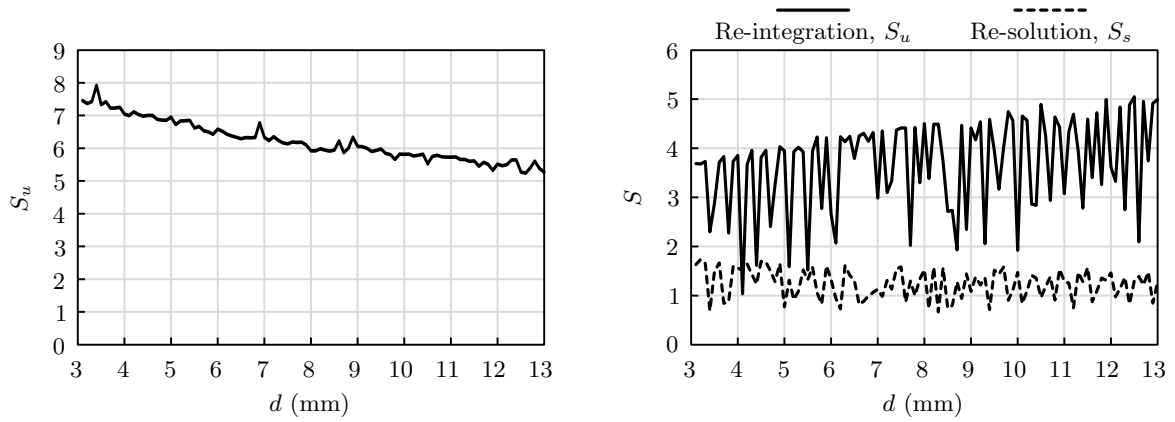
(c) Accelerated re-analysis on eight processors.

Figure 10.6: Run time for a countersunk hole.

tional boundary element (BE) analysis, is shown in Figure 10.7. On average the re-integration speed-up, $S_u = 6.2$ when applying the acceleration schemes suggested in this work on a single processor. However, as shown in Figure 10.7(a), S_u is greatest for the initial model and decreases as d increases. This is because the geometric change causes elements around the updated geometry to be distorted such that they are closer together. The influence of these elements on each other is thereby increased and a higher order integration scheme must be applied to capture the stress variation. Note that the re-solution speed-up, S_s, achieved by applying the GMRES scheme instead of a direct solver is not shown in Figure 10.7(a) as $S_s \approx 45$.

The algorithm is accelerated further through parallel processing, achieving an additional speed-up of $S_u \approx 3.7$ on eight processors over the serial implementation, as shown in Figure 10.7(b). Combining this with the results from Figure 10.7(a) gives a total average acceleration of $S_u \approx 22.9$ over a traditional serial BE implementation and $S_s \approx 58.1$ over a serial direct solver. The total speed-up of the combined re-integration and re-analysis algorithms over a full analysis with a direct solver, $S \approx 42.6$.

By utilising the techniques introduced in this work it becomes possible to re-analyse the countersunk hole model twice every second. This is sufficient to provide the analyst with useful feedback during dynamic updating of the model. However, a faster analysis is desirable to reduce flicker in the contour plot as the model is updated. A further reduction in t could be achieved by using a coarser mesh. However, this would result in a reduction in the quality of the stress results. Once the model has been finalised, a more detailed mesh and analysis may be employed to validate the design and provide more accurate results.



(a) Re-integration speed up through acceleration schemes.

(b) Speed up on eight processors in addition to that given in (a).

Figure 10.7: Re-analysis speed-up for a countersunk hole.

10.3 Case study: Aircraft rib section

10.3.1 The model

The model used in this case study is described in Figure 10.8. This model has been developed, in conjunction with industrial partners, to simulate a small section from an aircraft wing rib. The roller boundary conditions applied to the model indicate lines of symmetry. A nominal load of 10MPa is applied as indicated. The rib section has been modelled using 1290 elements with 7500 DOF.

To find the optimum radius, r, for the fillet indicated in Figure 10.8, r is initially set to 1.5mm which is the thickness of the step. The radius is then incrementally increased to establish how this affects the stress distribution. Data have been collected on all aspects of the algorithm; the results are summarised in this section.

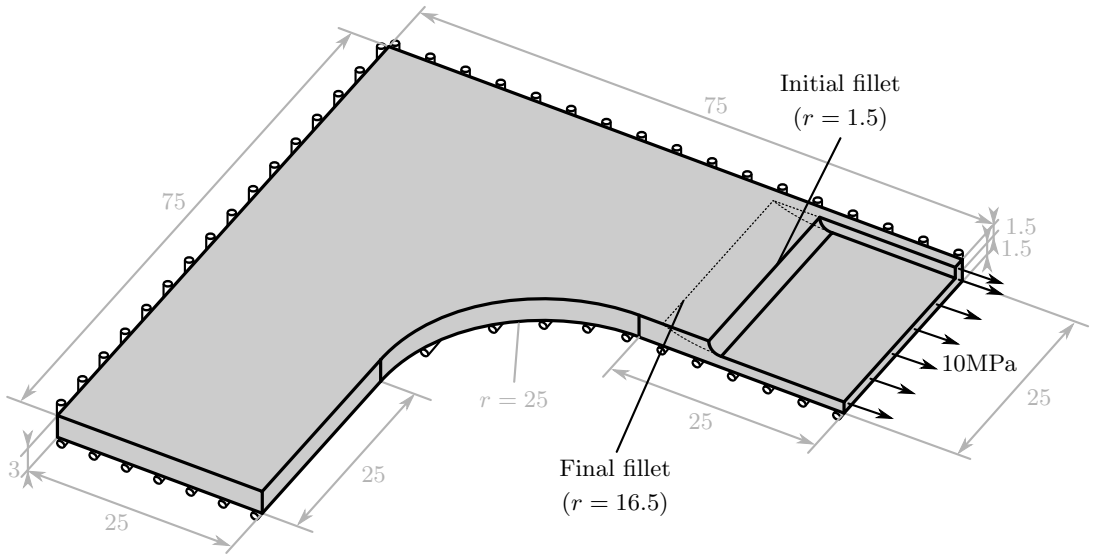
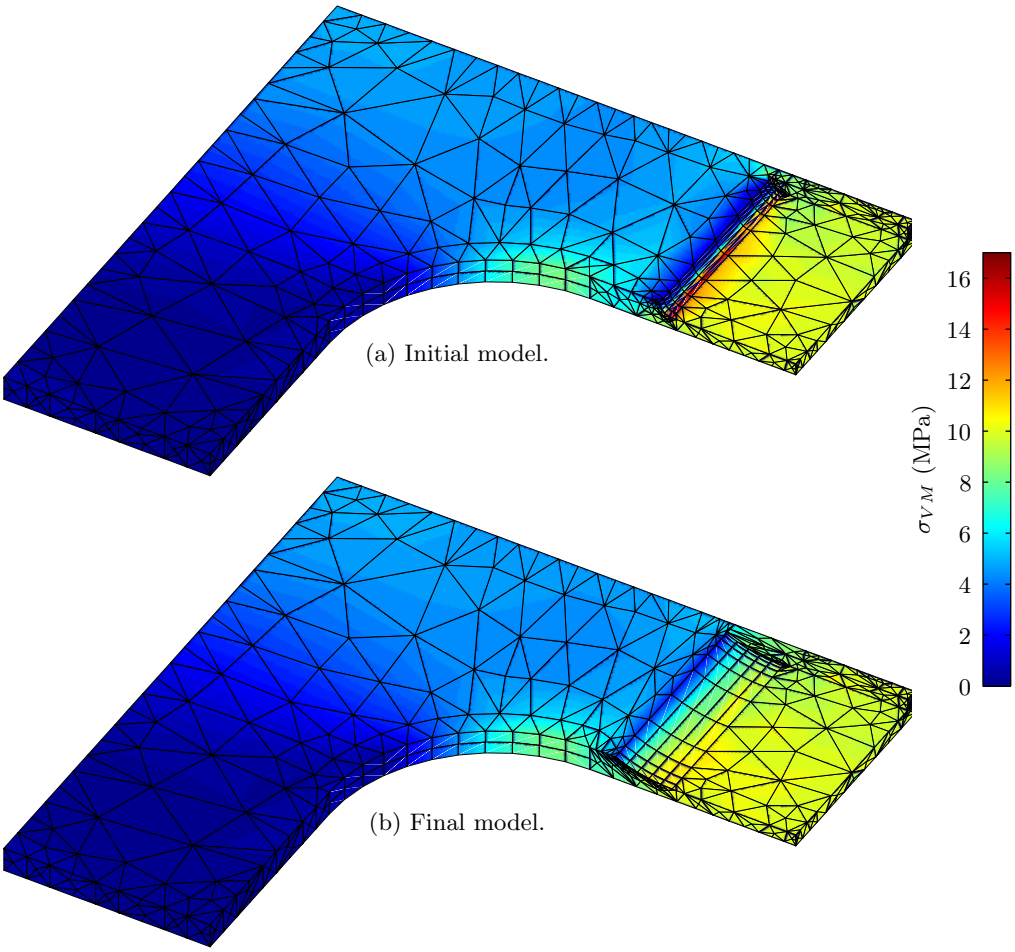


Figure 10.8: Case study: Aircraft rib section. (Dimensions in mm)

10.3.2 Results

The stress, σ_{VM}, across the surface of the rib section is shown in Figure 10.9. A stress concentration, $K_t = 1.67$, is apparent in the fillet on the initial model ($r = 1.5$mm). This reduces as r is increased to 16.5mm as shown in Figure 10.10. The results across the rest of the model remain almost constant. The stress concentration in this fillet should continue to reduce until $K_t = 1.0$. However, due to interactions with the other fillet in the model, K_t reaches a minimum of $K_t = 1.28$ when $r = 5.6$mm before increasing slightly. The aim of the analyst is to minimise K_t at the fillet, this would be apparent as the point at which the contours ceased to show a significant reduction in the stress.



(a) Initial model.

(b) Final model.

Figure 10.9: von Mises stress concentration over an aircraft rib section.

It should be noted that the contour plots in Figure 10.9 have been created by linear interpolation between the values of stress at the nodes. It is therefore imperative that the contour results are treated with caution. The values at the nodes, however, can be taken at face value within appropriate error bounds.

Figure 10.11 shows the distribution of element quality, Q, on the initial mesh ($r = 1.5$mm) and the final mesh ($r = 16.5$mm). The initial mesh is of high quality with the exception of a few elements along the narrow sections of the geometry. This is caused by limitations placed on the meshing scheme so as not to over refine the mesh. It should also be noted that, whilst some of the rectangular elements on the fillet have a low Q, due to their orientation relative to that of the stress field, this does not affect the accuracy of the results.

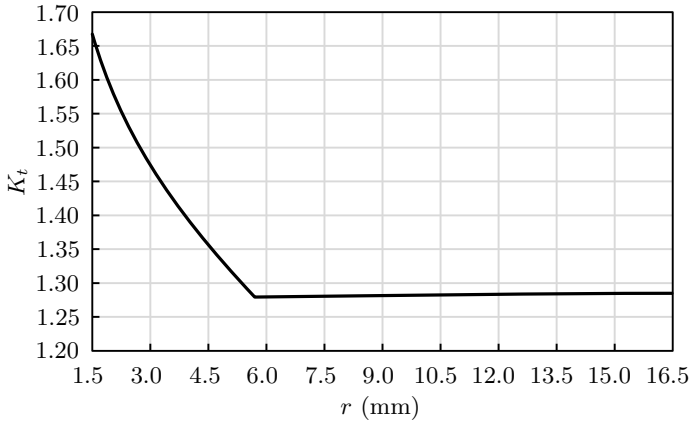


Figure 10.10: Stress concentration factor at the fillet.

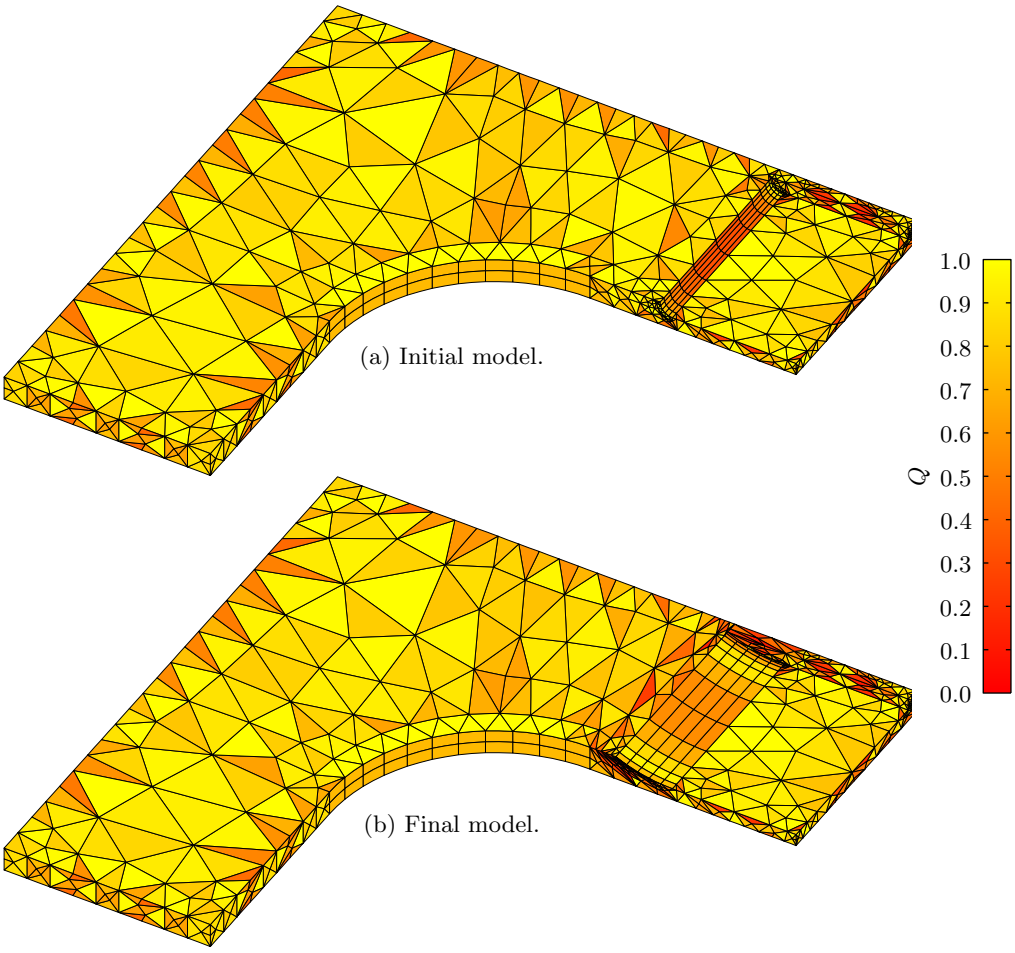
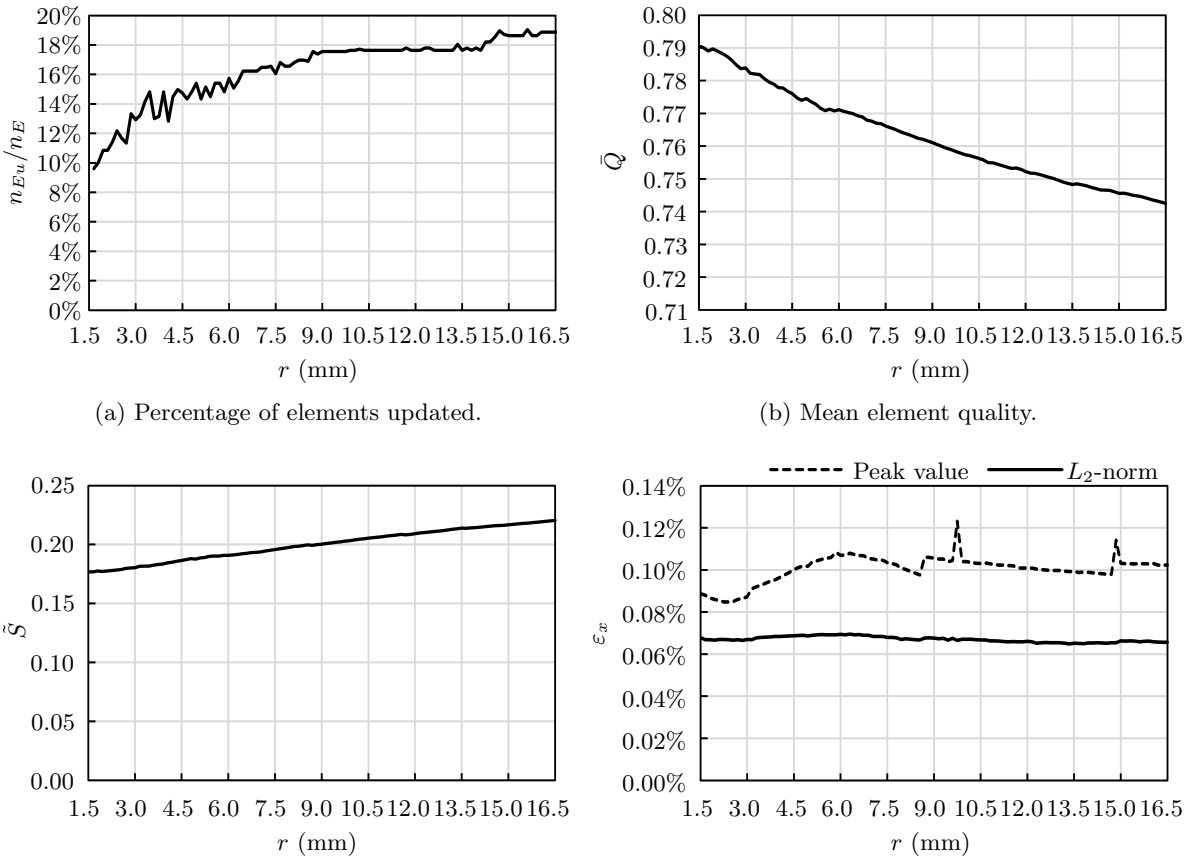


Figure 10.11: Element quality over an aircraft rib section.

As the geometry is updated, the majority of the mesh remains the same. However, a degradation in the quality of elements around the updated geometry can clearly be seen. If r were to be increased further, local re-generation of the mesh would be required to maintain a suitable element quality. The mean element quality is given in Figure 10.12(b) and remains within acceptable limits. The increase in error, ε_x, shown in Figure 10.12(d), caused by applying the acceleration schemes is approximately constant for all values of r and remains small.



(a) Percentage of elements updated.

(b) Mean element quality.

(c) Standard deviation of element quality.

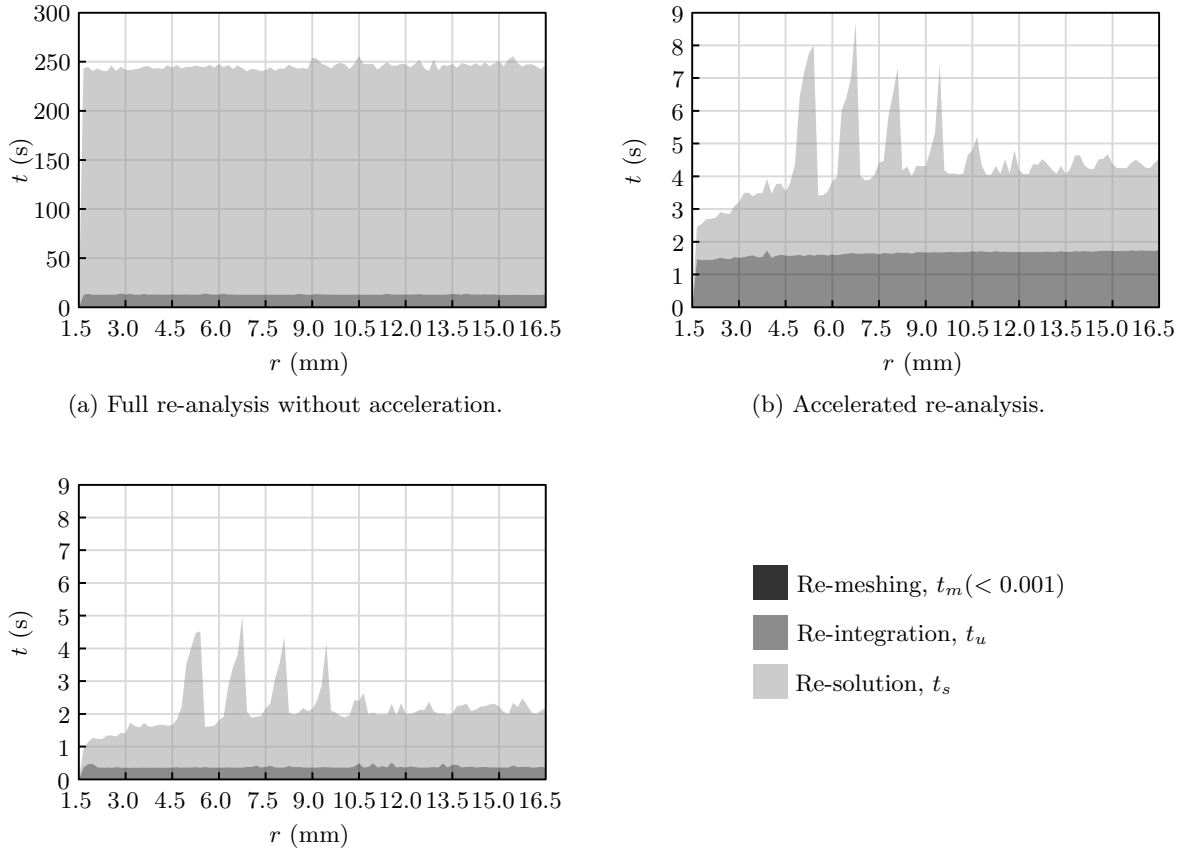
(d) Increase in error, ε_x, over re-integration without acceleration schemes.

Figure 10.12: Mesh and re-meshing results for an aircraft rib section.

10.3.3 Timings

The distribution of re-integration time, t, across re-meshing (t_m), re-integration (t_u) and re-solution (t_s) as r is increased is shown in Figure 10.13. Figure 10.13(a) shows t using a full re-integration with a direct solver. An even higher proportion of t is dedicated to t_s than for Case Study 1. This is due to the larger number of DOF in Case Study 2. However, because of the larger problem size, iterative solvers prove more effective. For this model, applying the LU preconditioned GMRES solver reduces t_s by a factor of 250 over a Gauss elimination. If this reduced solve time is added to the accelerated re-integration, Figure 10.13(b) is produced. However, due to the problem size, t_s still dominates. This dominance is increased further when the algorithm is parallelised, as shown in Figure 10.13(c).

The time to re-mesh the model, t_m is less than 0.001 seconds and is constant for all three figures. It is included in Figure 10.13 but is too small to be apparent. The meshing and re-meshing schemes have not been parallelised as there would be little benefit to accelerating this algorithm further.



(a) Full re-analysis without acceleration.

(b) Accelerated re-analysis.

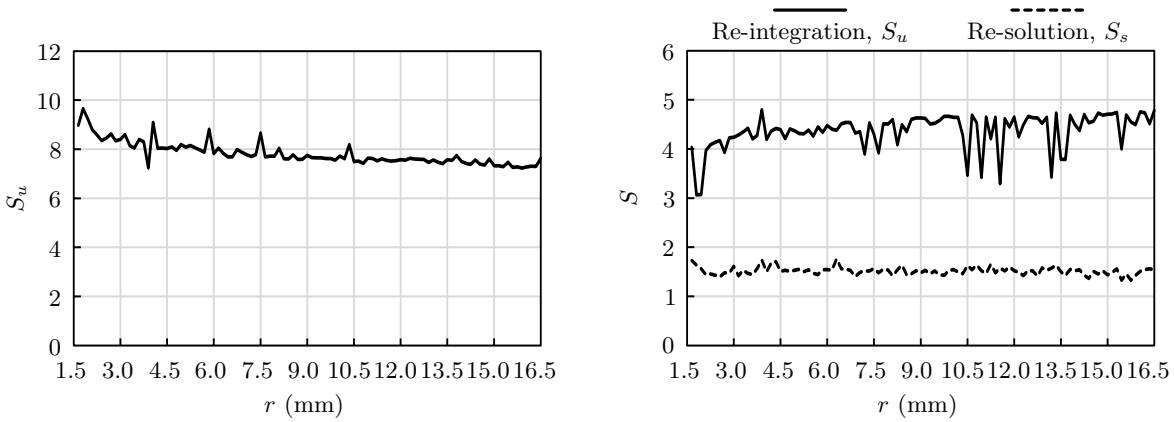
(c) Accelerated re-analysis on eight processors.

Figure 10.13: Run time for an aircraft rib section.

The speed-up, S, achieved by the new re-integration and re-solution schemes is shown in Figure 10.14. An average speed-up of $S_u = 7.8$ over a traditional BE integration is achieved when applying the acceleration schemes suggested in this work on a single processor. This is larger than for the countersunk hole because the larger model (in terms of DOF) results in an increase in the number of far field elements paired with each node in the model. These far field elements only require a coarse integration scheme and ar therefore fast to integrate. As with Case Study 1, S_u decreases as the element distortion increases due to the closer proximity of distorted elements to each other. Note that the re-solution speed-up, $S_s \approx 250$, for the GMRES scheme over a direct solve is not shown in Figure 10.14(a).

When the algorithm is parallelised, an additional speed-up of $S_u \approx 4.4$ is achieved on eight processors, as shown in Figure 10.14(b). This is greater than for the countersunk hole model as the larger number of DOF results in a more efficient parallelisation. Combining these results with those from Figure 10.14(a) gives a total average acceleration of $S_u \approx 34.3$ over a traditional serial BE implementation and $S_s \approx 73.8$ over a serial direct solver. The total speed-up of the combined re-integration and re-analysis algorithms over a full analysis with a direct solver, $S \approx 62.7$.

The total analysis time, t, for the aircraft rib section has been significantly reduced by employing the acceleration schemes developed in this work. However, for a model with 7500 DOF, this is not sufficient for real-time analysis although rapid feedback is still provided with a typical total re-analysis time in the region of 2.2 seconds per iteration.



(a) Re-integration speed up through acceleration schemes.

(b) Speed up on eight processors in addition to that given in (a).

Figure 10.14: Re-analysis speed-up for an aircraft rib section.

10.4 Software implementation

10.4.1 The philosophy behind the software

This section describes the new computer aided design (CAD) software package CA3D, named after Concept Analyst (CA) [1] (a two-dimensional package with the same design concept), which the algorithms in this work have been designed for use with. The main consideration is the GUI which has been primarily developed by a colleague of the author, M. Shadi Mohamed. To enable ease and speed of design, the GUI has been kept to a minimalist form which requires as few user inputs as possible. A screenshot of the programme is shown in Figure 10.15. All the geometry is constructed using the Open Cascade [166] modelling libraries.

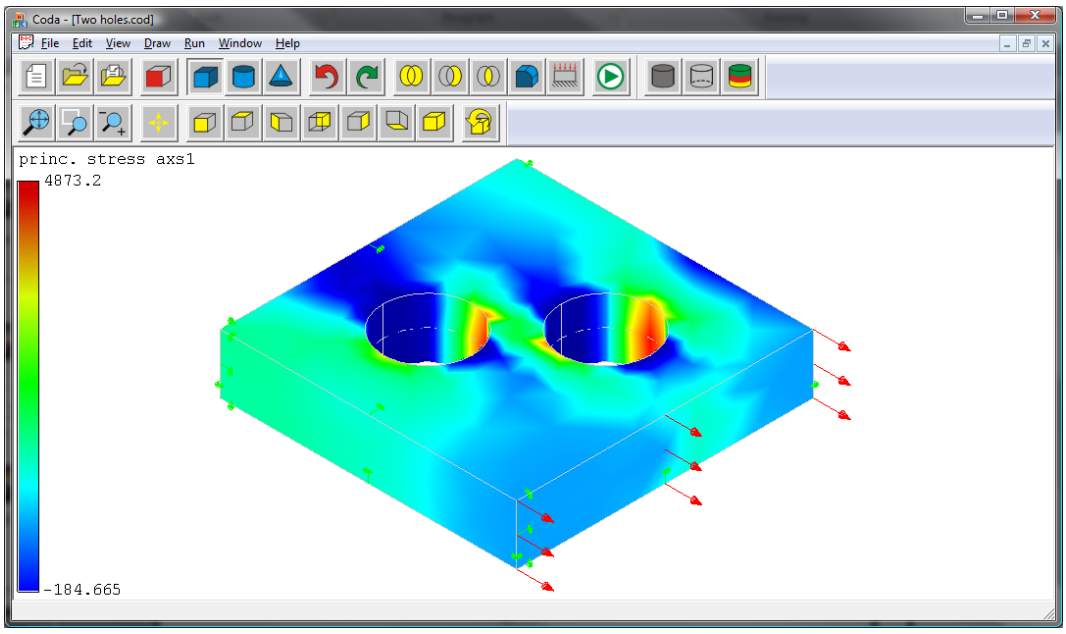


Figure 10.15: Concept Analyst 3D.

10.4.2 The graphical user interface

The CA3D toolbar is shown in Figure 10.16. This contains the main commands which are used to construct and analyse a model.

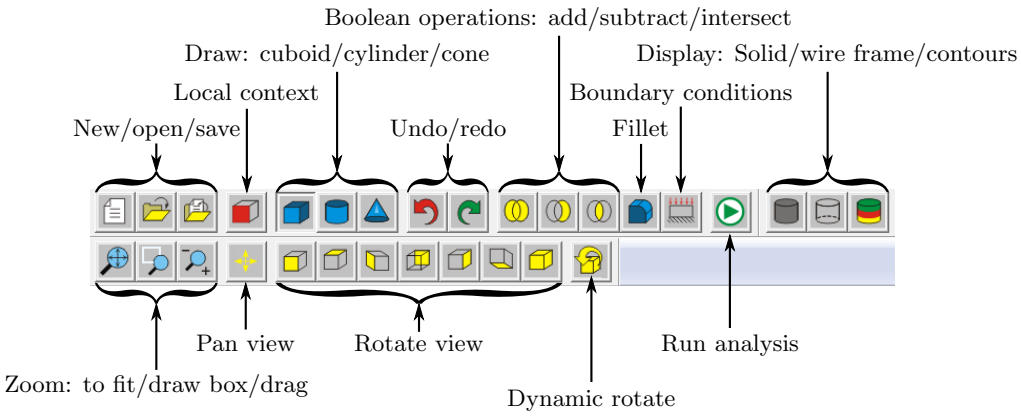
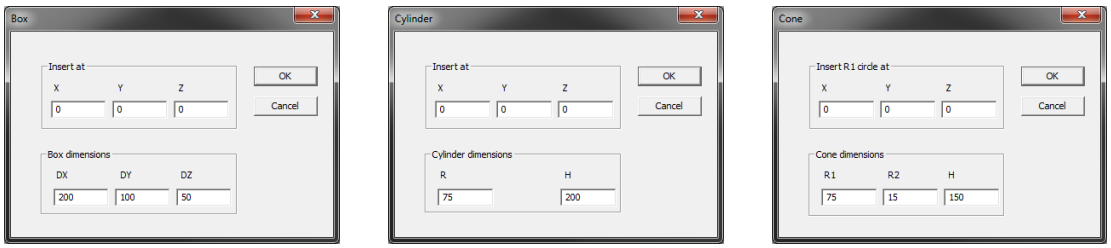


Figure 10.16: The toolbars.

The 'draw' dialogue boxes are shown in Figure 10.17. These incorporate the (x, y, z) coordinates of the object, which define the minimum coordinate location of the corner of a cuboid or the axis of rotation of a cylinder or cone. The dialogue allows the dimensions to be specified. For the cuboid, 'DX', 'DY' and 'DZ' refer to the (x, y, z) dimensions of the object; for the cylinder, 'R' and 'H' refer to the radius and length respectively; for the cone, 'R1' and 'R2' refer to the radius of each end, with 'R1' at the insertion point. The radius of a cone may be set to zero should a complete cone be required.

The cylinder and cone insertion is currently limited in that they can only be created parallel to the z axis. This has been limited to reduce the possibility of creating illegal geometries through intersecting cones and cylinders with axes of rotation at different angles.



(a) Cuboid. (b) Cylinder. (c) Cone.

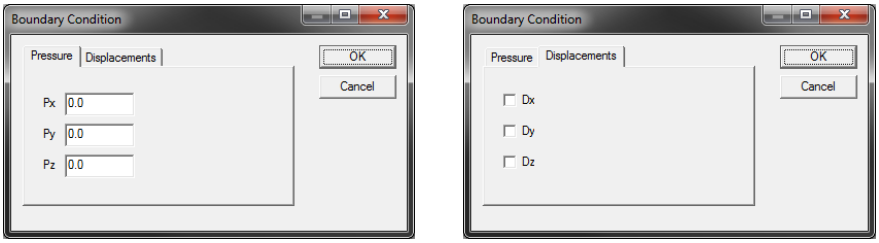
Figure 10.17: 'Draw' dialogue boxes.

Three different Boolean operations may be applied to the geometry by the user. 'Add' will merge two selected objects into a single object and 'subtract' will remove the the volume denoted by the second object selected by a user from the first object selected. 'Intersect' will remove all material except for where the two objects overlap. The 'fillet' dialogue allows the user to specify a radius for a fillet, as shown in Figure 10.18. The fillet must be applied to an edge or a group of edges. If the radius is greater than the size of a neighbouring face, a partial fillet will be applied.

The 'boundary conditions' dialogue, shown in Figure 10.19 allows the user to specify the pressure applied to, or to fix the displacement of, a face. Additional functionality is available to load or restrain surfaces in the normal direction or apply displacements. However, at the time of writing these have not

been incorporated into the user GUI. The boundary conditions are displayed on the model as shown in Figure 10.15. Pressures are shown as red arrows, restraints as green anchor points.

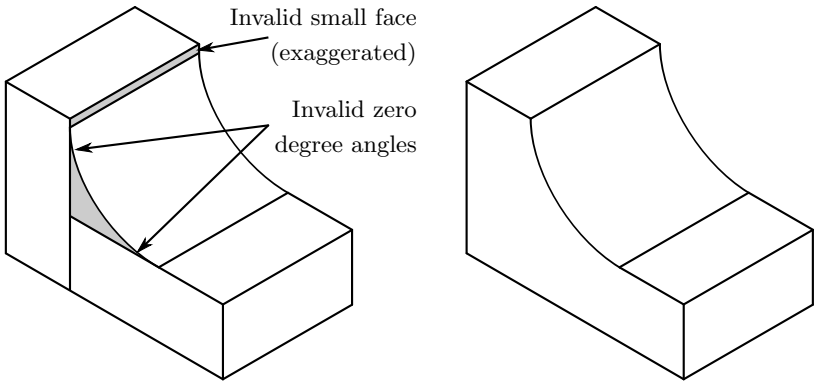
Figure 10.18: 'Fillet' dialogue box.



(a) Pressure. (b) Displacement.

Figure 10.19: 'Boundary conditions' dialogue box.

The 'run' command runs a full initial analysis on the model. This generates and stores all the data associated with the integration of each element along with the $[H]$ and $[G]$ matrices which are referred to in the re-analysis. A direct solver is used to form the LU preconditioner. An additional one-shot analysis is available which directly constructs $[A]$ and $\{b\}$ without forming the complete $[H]$ and $[G]$ matrices. The additional integration data is not stored and, as the preconditioner will not be required, a diagonally preconditioned GMRES solver is applied. The one-shot analysis is therefore significantly faster than the full initial analysis. However re-analysis cannot be carried out if this option is selected. When the model is run, algorithms will be applied to automatically merge co-planar faces and remove any small faces. Co-planar faces are merged to ensure invalid geometries are not meshed, such as the case of a zero angle, illustrated in Figure 10.20. The 'remove small faces' algorithm attempts to remove tiny faces generated by the Open Cascade fillet application process. These faces would otherwise cause massive over refinement of the mesh, leading to an invalid mesh. Both algorithms may also be run independently, via the menu system, without triggering an analysis.



(a) Before clean up. (b) After clean up.

Figure 10.20: Removing invalid geometry.

Contours may be displayed over the surface of the model. These are drawn by interpolating the stresses across each element from the values at the nodes with bright red as the highest value and dark blue the lowest value. An option is also available to display the deformed model. The 'contour plot' dialogue allows the user to specify the results they wish to plot:

- Principal stresses $(\sigma_1, \sigma_2, \sigma_3)$
- Direct stresses $(\sigma_{xx}, \sigma_{yy}, \sigma_{zz})$
- Shear stresses $(\sigma_{xy}, \sigma_{yz}, \sigma_{xz})$
- von Mises stress (σ_{VM})
- Displacements $(\delta_x, \delta_y, \delta_z)$

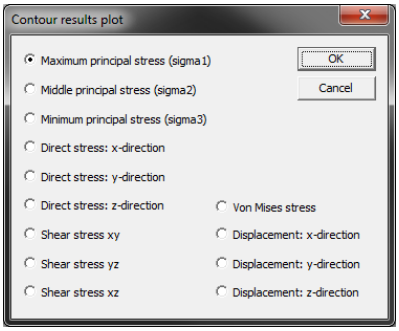


Figure 10.21: 'Contour plot' dialogue box.

In addition to the buttons on the toolbar, several other options can be accessed via the menus. The material properties and the mesh density may be changed via the 'material properties' dialogue shown in Figure 10.22. For simplicity, three options are given for the mesh density: coarse, medium and fine.

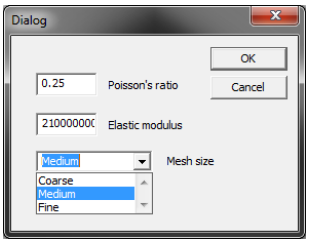


Figure 10.22: 'Material properties' dialogue box.

The model geometry may be updated by selecting a face to update in the local context, then using the 'move faces' command before clicking and dragging the selected face. During dynamic updating of the geometry, movement of cylinders and cones is limited to motion perpendicular to the axis of rotation. Plane faces may only be moved in a direction which is normal to the face. This helps to reduce the propagation of changes through the mesh by limiting the area of the model which the updates affect. Whilst the geometry is being updated, the stress contours that will continue to be displayed will be of the same type as those last selected using the 'contour plot' dialogue.

If no Boolean operations have been carried out but more than one geometrical construct has been formed then multiple objects will exist in the model. Selecting the 'local context' button allows the user to select faces, edges and vertices within the currently selected object. In the local context, the default selection mode is 'faces'. This may be changed to 'vertices' or 'edges'. Using 'faces', boundary conditions can be applied to the model. Selecting 'vertices' will allow the user to read off the coordinates of the selected vertex. Selecting 'edges' will allow the user to apply fillets.

Chapter 11

Conclusion

"I may not have gone where I intended to go, but I think I have ended up where I intended to be."

Douglas Adams

11.1 Conclusions

The algorithms developed in this thesis demonstrate that it is possible to conduct real-time re-analysis of the stresses in three-dimensional linear elastic boundary element (BE) components as the geometry of these components is dynamically updated by the user. This was achieved by applying an intelligent re-meshing scheme capable of accommodating a geometric perturbation whilst preserving the majority of the mesh and hence, limiting changes in the system of equations associated with the problem. This in turn leads to a reduction in the time required to construct these equations. The re-meshing scheme is highly efficient for small perturbations, typically taking a few thousandths of a second, around six times faster than generating the initial mesh. Methods to improve the scheme for larger changes have also been discussed.

The reconstruction of the system of equations has been accelerated through the use of adaptive integration schemes (incorporating a reusable intrinsic sample point (RISP) algorithm) and computational tactics such as reducing the precision of the calculations and using 64-bit architecture. Whilst parallelising the process on central processing units (CPUs) showed good speed-up and scalability, the system is still too small for graphics processing unit (GPU) based acceleration to be useful. Adaptive cross approximation (ACA) was found to be inefficient for models with relatively few degrees of freedom (DOF), such as those encountered in this work.

Several different linear solvers have been assessed to establish the most applicable to the small dense matrices typical of the boundary element method (BEM). These tests established that a generalised minimal residual (GMRES) solver used in conjunction with a complete approximate lower-upper (LU) preconditioner, generated during the first analysis run before any geometric modifications had been applied, provided the fastest and most accurate solution. Minimising the number of updated elements in the system will also help to reduce the solve time as the preconditioner will be more similar to the inverse of the new system and hence the iterative solver will show a faster convergence rate. A small speed gain was made by parallelising the GMRES solver on CPUs. Whilst GPU parallelisation showed good scalability, it was found not to be currently capable of accelerating the solution for models with fewer than 5000 DOF.

The algorithms created through this work have been comprehensively tested using several different example problems of particular interest in the aerospace industry. They have been shown to be fast,

robust and accurate, paving the way for real-time re-analysis of BE problems. A new three-dimensional BE analysis package, Concept Analyst 3D (CA3D), capable of real-time re-analysis of basic mechanical components as the geometry of these components is dynamically updated by the user, has been developed using the techniques demonstrated in this thesis.

The future of computational mechanics will involve the evolution of simulations alongside model geometry. Engineers will be naturally and seamlessly presented with key durability information as they construct a design and will therefore be able to adjust the design, based on these considerations, as it evolves. The work presented in this thesis contributes towards this vision.

11.2 Recommendations for further work

Recommendations for further work have been divided up into four sections. Section 11.2.1 comments on features that are available in the meshing and analysis algorithms but are not supported by the graphical user interface (GUI) at the time of writing. Section 11.2.2 looks at ways to improve the current method and algorithms. Section 11.2.3 discusses how the method could be adapted to address different types of problem. Section 11.2.4 looks at the future evolution of the computer and how this could affect further development of the ideas presented in this work.

11.2.1 Additions to the Concept Analyst 3D software package

At the time of writing, the GUI only permits cones and cylinders to be generated such that the axis of rotation is parallel to the z axis. This constraint has been imposed to ensure that the geometry does not intersect in ways that cannot currently be meshed. However, the meshing algorithm supports geometry in any orientation as long as it conforms to the supported types given in Table 11.1. If the additional geometries given in Section 11.1 were included it would become possible to mesh more complex intersecting geometries. The analysis algorithms support any valid BE mesh constructed of trianglar and quadrilateral elements.

An additional option has been included in the boundary conditions class that allows boundary conditions to be declared in the normal direction to a surface. This is particularly important for application across curved surfaces such as a thick walled cylinder under an internal pressure. This option is not currently available in the GUI as additional calculations would be required to orientate the arrows showing the pressure on the model. The GUI also does not allow non-zero displacements to be applied to the model.

During dynamic updating of the model geometry, the GUI limits translation of cylindrical and conical surfaces to movement perpendicular to the axis of rotation. It is not possible to dynamically change the depth or angle of a countersink or the radius of a hole or fillet. However, these operations are supported by the re-meshing scheme and could be added to the GUI at a later date. It should be noted that the case studies given in Chapter 10 bypass the movement constraints applied in the GUI.

Rotation of patches and elements without changing the local geometry is currently classed as a distortion. Functionality could be added to the code so that this type of rotation could be treated in a similar way to a translation.

11.2.2 Improving the current method

The majority of improvements to the current method can be made in the meshing algorithm. The current algorithm is sufficient to demonstrate that real-time re-analysis can be achieved. However, it imposes many constraints on the geometries that can be analysed. Additional geometries would be required to make the software viable for general design work. The currently supported geometries and the proposed initial extensions are shown in Table 11.1. The addition of elliptical arcs and parabolas would allow

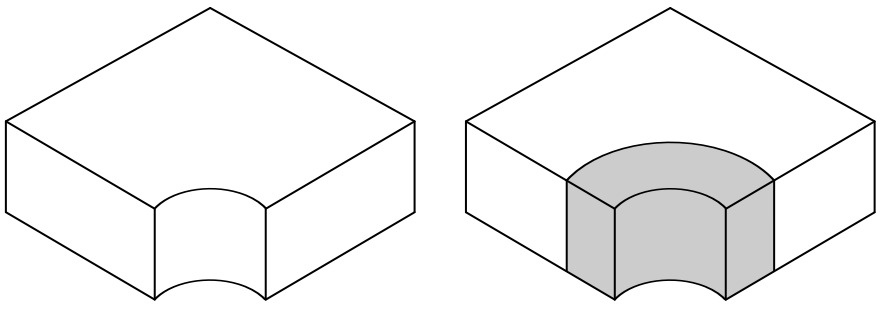
cylindrical and conical surfaces to be cut at an angle. Toroids would allow for toroidal fillets and spheres would allow three or more fillets to meet at a vertex. If the method were to be extended to spline surfaces a complete re-structuring of the re-meshing algorithm would be required as the geometric changes applied to the model would not be so easy to isolate.

Table 11.1: Supported geometry.

	Currently supported	**Proposed additions**
2D	Straight line Circular arc	Elliptical arc Parabola
3D	Plane surface Cylinder Cone	Sphere Torus

The mesh generation strategy does not take into account the boundary conditions. To generate an even more efficient mesh these should be considered in the initial mesh generation and the re-meshing procedure. A mesh refinement algorithm could be incorporated into the re-meshing procedure to refine the mesh around high stress concentrations as the user updates the model. However, such an algorithm should be employed sparingly as running constantly will have a direct impact on the speed of the re-meshing algorithm and cause more elements to be updated, thereby slowing the re-analysis.

More advanced feature recognition could be employed to identify areas of geometry the user is likely to update. Alternatively the user could be asked when they initially construct the model to define which areas they are particularly interested in. This data could be used to inform the meshing and re-meshing scheme so that the mesh can be appropriately refined. One technique that could be employed is zoning. Zoning involves splitting the model into sub-regions for the purposes of meshing and analysis without affecting the user's perception of the model. Areas of the model around features that are likely to be updated by the user, such as the radius of a hole, can be defined using a separate volume and set of patches to the rest of the model as shown in Figure 11.1. If the feature is updated the mesh can simply be regenerated over the appropriate zone, leaving the majority of the model unchanged and removing the need to re-run analyses over large portions of the structure. However, zoning will introduce additional nodes to the model across the interface patch, which will result in a longer initial run time, but it can improve the quality of the system matrix, which will counteract this to some extent by helping to reduce the analysis time. Zoning must be used if there are multiple materials present within the model.



(a) Single zone. (b) Two zones.

Figure 11.1: Zoning around features.

11.2.3 Extensions to Concept Analyst 3D

This work has concentrated on simple, elastic stress problems. The algorithms could easily be modified to cover potential problems and acoustics or waves. Additions could also be made to solve fracture problems and, with the inclusion of real-time solution, it would be possible to produce instant feedback on simulations of crack growth problems, thereby giving the analyst more detailed feedback on the path and speed of growth of the crack.

It would also be useful to be able to simulate contact problems. The case of a countersunk hole has regularly surfaced in this work. However, this has always assumed an empty hole. It would be useful to be able to model the effect of putting a screw into the hole.

11.2.4 Future considerations

The size of problem that can be solved in real-time using the techniques discussed in this work has been limited by the processing capability of current computers. This capability is constantly improving and evolving. It will not be many years before it becomes possible to apply these techniques to solve much larger BE problems in real-time. However, as the problem size increases other acceleration techniques will become viable.

ACA has been shown to work well for larger matrix blocks which result from a larger system. Similarly, proper orthogonal decomposition (POD) will provide more benefits as the problem size increases. There are other techniques, not considered in this work, which will also become viable as the problem size increases.

The parallel implementation of the integration scheme has shown good scalability. As the number of CPUs in standard computers increases, the speed of the algorithm will also increase. However, more efficient management load balancing of each CPU could lead to increased benefits, especially as the number of CPUs increases.

So far it has only been possible to solve small problems in real-time and GPUs rely on massively parallel systems. Current GPU capability has been shown to only accelerate the solution of linear problems of size $n > 5000$. The main limiting factor is the memory transfer speed between the GPU and the CPU. As GPUs evolve the memory speed will increase. Some early CPU designs where the GPU cores are built directly onto the processor are already in development. However, increased speed will also lead to the ability to solve larger problems which are more susceptible to GPU acceleration.

Isogeometric analysis [171] is currently emerging as a tool for analysing models directly from computer aided design (CAD) geometry. The BEM is ideally suited to exploit this technique as CAD geometries are generally constructed using a surface representation [172, 173]. Incorporating the techniques developed in this thesis into an isogeometric analysis package would further facilitate the visualisation of stresses over CAD geometry as a model is being constructed without the need for meshing and re-meshing. The development of isogeometric analysis will pave the way for full integration of CAD and analysis packages into a single modelling environment.

11.3 Applications

This work has been conceived to create an interactive tool to enable engineers to dynamically adjust the design of mechanical components based on the stress results of an analysis. However, this work could be applied in many other fields.

Shape and topology optimisation algorithms all rely on analysis of a sequence of models of evolving geometry. These algorithms are ideally suited to take advantage of the techniques developed in this thesis. It may eventually be possible to cut out the designer almost entirely, as the geometry of the

model is automatically updated by the computer to find the best possible form for the component under investigation. If an optimisation algorithm were coupled to real-time analysis it would become possible to observe the geometry as it evolved and perhaps steer the results where necessary.

Real-time analysis in the context of haptic feedback for surgical simulation has already been extensively discussed in this work. Haptic feedback requires a particularly high frequency response rate of around 1kHz and this work is not yet able to provide sufficient speed. However, the need for such accurate stress solutions is not so pressing and a coarser model and integration scheme could provide suitably accurate displacement results to ensure the simulation feels realistic. During surgery, if surgeons need to change their procedure, real-time analysis could enable them to instantly see what will happen and aid their decision making.

For pure visualisation of the displacements, the current algorithm is more than adequate. The techniques employed here could therefore be incorporated into computer games to provide real-time visual feedback as the player interacts with objects in the virtual environment.

If a structure were already undergoing some form of ongoing deformation or cracking, the real-time techniques discussed in this work could be used to rapidly analyse the current situation and project what would be expected to happen as it developed, before the structure actually reached this state. This could provide a useful prediction tool for what to expect in damaged structures as time passes or the conditions around them change.

11.4 Summary

A new set of algorithms for accelerating three-dimensional BE re-analysis of basic mechanical components with continuously updating geometry has been introduced. This makes it possible to analyse small models (in terms of DOF) in real-time and significantly accelerate analysis of larger models. Models with up to 4000 DOF can be consistently re-analysed in under half a second.

Whilst there are some drawbacks in the algorithms, such as the limited range of geometry that they are able to analyse, these can easily be overcome through further development of the algorithms. This work serves to prove that the methods employed to accelerate the analysis and re-analysis are robust and efficient and enable small models to be accurately analysed in real-time. As computer hardware develops it is expected that much larger time savings can be made for larger models that it is currently impossible to analyse in real-time.

This work has paved the way for real-time three-dimensional BE analysis, demonstrating that it is a viable approach and that it can be implemented in CAD software acceptable for use in industry.

References

[1] "Concept Analyst Ltd." www.conceptanalyst.com, 2013.

[2] W. D. Pilkey, *Peterson's Stress Concentration Factors (Second Edition)*. John Wiley & Sons, 1997.

[3] J. H. Kane, B. L. K. Kumar and R. H. Gallagher, "Boundary-element iterative reanalysis for continuum structures", *Journal of Engineering Mechanics*, vol. 116(10), pp. 2293–2309, 1990.

[4] R. Mackie, "An object-orientated approach to fully interactive finite element software", *Advances in Engineering Software*, vol. 29, pp. 139–149, 1998.

[5] M. J. Ryken and J. M. Vance, "Applying virtual reality techniques to the interactive stress analysis of a tractor lift arm", *Finite Elements in Analysis and Design*, vol. 35, pp. 141–155, 2000.

[6] L. Margetts, C. Smethurst and R. Ford, "Interactive finite element analysis", in *NAFEMS World Congress, Malta*, 2005.

[7] S. S. Terdalkar and J. J. Rencis, "Graphically driven interactive finite element stress analysis for machine elements in the early design stage", *Finite Elements in Analysis and Design*, vol. 42, pp. 884–899, 2006.

[8] U. Meier, O. López, C. Monserrat, M. C. Juan and M. Alcañiz, "Real-time deformable models for surgery simulation: A survey", *Computer Methods and Programs in Biomedicine*, vol. 77, pp. 183–197, 2005.

[9] P. Wang, A. A. Becker, I. A. Jones, A. T. Glover, S. D. Benford, C. M. Greenhalgh and M. Vloeberghs, "A virtual reality surgery simulation of cutting and retraction in neurosurgery with force-feedback", *Computer Methods and Programs in Biomedicine*, vol. 84, pp. 11–18, 2006.

[10] P. Wang, A. A. Becker, I. A. Jones, A. T. Glover, S. D. Benford, C. M. Greenhalgh and M. Vloeberghs, "Virtual reality simulation of surgery with haptic feedback based on the boundary element method", *Computers and Structures*, vol. 85, pp. 331–339, 2007.

[11] M. Bro-Nielson and S. Cotin, "Real-time volumetric deformable models for surgery simulation using finite elements and condensation", *Computer Graphics Forum*, vol. 15(3), pp. 57–66, 1996.

[12] D. James and D. Pai, "ArtDefo - Accurate real time deformable objects", in *SIGGRAPH 99, Los Angeles, USA*, pp. 65–72, 1999.

[13] S. F. F. Gibson and B. Mirtich, "A survey of deformable modeling in computer graphics", *Technical report: Mitsubishi Electric Research Laboratories, Cambridge, MA*, vol. TR-97-19, 1997.

[14] E. Kita and N. Kamiya, "Error estimation and adaptive mesh refinement in boundary element method, an overview", *Engineering Analysis with Boundary Elements*, vol. 25, pp. 479–495, 2001.

[15] K. F. C. Yiu, D. M. Greaves, S. Cruz, A. Saalehi and A. G. L. Borthwick, "Quadtree grid generation: Information handling, boundary fitting and CFD applications", *Computers and Fluids*, vol. 25(8), pp. 159–169, 1996.

[16] J. Trevelyan, P. Wang and S. K. Walker, "A scheme for engineer-driven mechanical design improvement", *Engineering Analysis with Boundary Elements*, vol. 26, pp. 425–433, 2002.

[17] J. Trevelyan and D. J. Scales, "Techniques to accelerate BEM computation to provide virtual reality update of stress solutions", *Engineering Analysis with Boundary Elements*, vol. 31, pp. 875–889, 2007.

[18] A. K. Michler, "Aircraft control surface deflection using RBF-based mesh deformation", *International Journal for Numerical Methods in Engineering*, vol. 88, pp. 986–1007, 2011.

[19] Y. Saad and M. Schultz, "GMRES: A generalised minimal residual algorithm for solving nonsymmetric linear systems", *SIAM Journal on Scientific Computing*, vol. 7, pp. 856–869, 1986.

[20] K. G. Prasad, J. H. Kane, D. E. Keyes and C. Balakrishna, "Preconditioned Krylov solvers for BEA", *International Journal for Numerical Methods in Engineering*, vol. 37, pp. 1651–1672, 1994.

[21] F. P. Valente and H. L. G. Pina, "Iterative techniques for 3D boundary element method systems of equations", *Engineering Analysis with Boundary Elements*, vol. 25, pp. 423–429, 2001.

[22] H. Xiao and Z. Chen, "Numerical experiments of preconditioned Krylov subspace methods solving the dense non-symmetric systems arising from BEM", *Engineering Analysis with Boundary Elements*, vol. 31(12), pp. 1013–1023, 2007.

[23] L. J. Leu, "A reduction method for boundary element reanalysis", *Computer Methods in Applied Mechanics and Engineering*, vol. 178, pp. 125–139, 1999.

[24] D. Amsallem and C. Farhat, "An interpolation method for the adaptation of reduced-order models to parameter changes and its application to aeroelasticity", *American Institute of Aeronautics and Astronautics Journal*, vol. 46, pp. 1803–1813, 2008.

[25] D. Ryckelynck, L. Hermanns, F. Chinesta and E. Alarćon, "An efficient 'a priori' model reduction for boundary element models", *Engineering Analysis with Boundary Elements*, vol. 29, pp. 796–801, 2005.

[26] P. Kerfriden, P. Gosselet, S. Adhikari and S. P. A. Bordas, "Bridging proper orthogonal decomposition methods and augmented Newton-Krylov algorithms: An adaptive model order reduction for highly nonlinear mechanical problems", *Computer Methods in Applied Mechanics and Engineering*, vol. 200, pp. 850–866, 2011.

[27] S. Marburg and S. Schneider, "Performance of iterative solvers for acoustic problems. part 1: Solvers and effect of diagonal preconditioning", *Engineering Analysis with Boundary Elements*, vol. 27, pp. 727–750, 2003.

[28] A. A. Becker, *The Boundary Element Method in Engineering.* McGraw-Hill International, 1992.

[29] J. H. Kane, *Boundary Element Analysis in Engineering Continuum Mechanics.* Prentice-Hall, 1994.

[30] J. Trevelyan, *Boundary Elements for Engineers: Theory and Applications.* Computational Mechanics Publications, 1994.

[31] Computational Mechanics, *BEASY user guide, volume 1*, 1998.

[32] G. R. Joldes, A. Wittek and K. Miller, "Real-time nonlinear finite element computations on GPU – Application to neurosurgical simulation", *Computational Methods in Applied Mechanics and Engineering*, vol. 199, pp. 3305–3314, 2010.

[33] A. H.-D. Cheng and D. T. Cheng, "Heritage and early history of the boundary element method", *Engineering Analysis with Boundary Elements*, vol. 29, pp. 268–302, 2005.

[34] M. A. Jaswon, "Integral equation methods in potential theory. I", *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 275, pp. 23–32, 1963.

[35] G. T. Symm, "Integral equation methods in potential theory. II", *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 275, pp. 33–46, 1963.

[36] F. J. Rizzo, "An integral equation approach to boundary value problems of classical elastostatics", *Quaterly of Applied Mathematics*, vol. 25, pp. 83–95, 1967.

[37] T. A. Cruse and F. J. Rizzo, "A direct formulation and numerical solution of the general transient elastodynamic problem. I", *Journal of Mathematical Analysis and Applications*, vol. 22, pp. 244–259, 1968.

[38] T. A. Cruse, "A direct formulation and numerical solution of the general transient elastodynamic problem. II", *Journal of Mathematical Analysis and Applications*, vol. 22, pp. 341–355, 1968.

[39] T. A. Cruse, "Numerical solutions in three dimensional elastostatics", *International Journal of Solids and Structures*, vol. 5(12), pp. 1259–1274, 1969.

[40] T. A. Cruse, "Application of the boundary-integral equation method to three dimensional stress analysis", *Computers and Structures*, vol. 3(3), pp. 509–527, 1973.

[41] T. A. Cruse, "An improved boundary-integral equation method for three dimensional elastic stress analysis", *Computers and Structures*, vol. 4, pp. 741–754, 1974.

[42] J. L. Swedlow and T. A. Cruse, "Formulation of boundary integral equations for three-dimensional elasto-plastic flow", *International Journal for Solids and Structures*, vol. 7, pp. 1673–1683, 1971.

[43] T. A. Cruse and J. L. Swedlow, "Interactive program for analysis and design problems in advanced composites technology", *United States Airforce Technical Report AFML-TR-71-268*, 1971.

[44] T. A. Cruse and W. van Buren, "Three-dimensional elastic stress analysis of a fracture specimen with an edge crack", *International Journal of Fracture Mechanics*, vol. 7, pp. 1–16, 1971.

[45] C. A. Brebbia and J. Dominguez, "Boundary element methods for potential problems", *Applied Mathematical Modelling*, vol. 1, pp. 372–378, 1977.

[46] T. S. A. Ribeiro, G. Beer and C. Duenser, "Efficient elastoplastic analysis with the boundary element method", *Computational Mechanics*, vol. 41, pp. 715–732, 2008.

[47] M. H. Aliabadi, *The Boundary Element Method: Applications in Solids and Structures*, vol. 2. John Wiley & Sons, 2002.

[48] M. Guiggiana, G. Krishnasamy, T. J. Rudolphi and F. J. Rizzo, "A general algorithm for then numerical solution of hypersingular boundary integral equations", *ASME Journal of Applied Mechanics*, vol. 59, pp. 604–614, 1992.

[49] G. T. Symm, "Boundary elements on a distributed array processor", *Engineering Analysis*, vol. 1, pp. 162–165, 1984.

[50] A. J. Davies, "Parallel implementations of the boundary element method", *Computers and Mathematics with Applications*, vol. 31(6), pp. 33–40, 1996.

[51] J. H. Kane, "Boundary element analysis on vector and parallel computers", *Computing Systems in Engineering*, vol. 5(3), pp. 239–252, 1994.

[52] B. A. Baltz and M. S. Ingber, "A parallel implementation of the boundary element method for heat conduction analysis in heterogeneous media", *Engineering Analysis with Boundary Elements*, vol. 19, pp. 3–11, 1997.

[53] K. Erhart, E. Divo and A. J. Kassab, "A parallel domain decomposition boundary element method approach for the solution of large-scale transient heat conduction problems", *Engineering Analysis with Boundary Elements*, vol. 30, pp. 553–563, 2006.

[54] N. Kamiya, H. Iwase and E. Kita, "Parallel implementation of boundary element method with domain decomposition", *Engineering Analysis with Boundary Elements*, vol. 18, pp. 209–216, 1996.

[55] N. Kamiya, H. Iwase and E. Kita, "Performance evaluation of parallel boundary element analysis by domain decomposition method", *Engineering Analysis with Boundary Elements*, vol. 18, pp. 217–222, 1996.

[56] M. Kreienmeyer and E. Stein, "Efficient parallel solvers for boundary element equations using data decomposition", *Engineering Analysis with Boundary Elements*, vol. 19, pp. 33–39, 1997.

[57] M. T. F. Cunha, J. C. F. Telles and A. L. G. A. Coutinho, "A portable parallel implementation of a boundary element elastostatic code for shared and distributed memory systems", *Advances in Engineering Software*, vol. 35, pp. 453–460, 2004.

[58] M. J. Stock and A. Gharakhani, "A GPU-accelerated boundary element method and vortex particle method", in *AIAA 40th Fluid Dynamics Conference and Exhibit, Chicago, USA*, 2010.

[59] C. V. Guillaume, "Introduction to GPGPU, a hardware and software background", *Comptes Rendus Mécanique*, vol. 339(2-3), pp. 78–89, 2011.

[60] A. Khamayseh and A. Kuprat, "Surface grid generation systems", in *Handbook of Grid Generation* (J. Thompson, B. Soni and N. Weatherill, eds.), ch. 9, CRC Press, 1999.

[61] B. H. V. Topping, J. Muylle, P. Ivanyi, R. Putanowicz and B. Cheng, *Finite Element Mesh Generation*. Saxe-Coburg, 2004.

[62] T. J. Baker, "Mesh generation: Art or science?", *Progress in Aerospace Sciences*, vol. 41, pp. 29–63, 2005.

[63] S. A. Canann, Y.-C. Liu and A. V. Mobley, "Automatic 3D surface meshing to address today's industrial needs", *Finite Elements in Analysis and Design*, vol. 25, pp. 185–198, 1997.

[64] D. A. Lavender and D. R. Hathurst, "An assessment of higher-order isoparametric elements for solving an elastic problem", *Computational Methods in Applied Mechanics and Engineering*, vol. 56, pp. 136–165, 1986.

[65] S. Yoshimura, Y. Wada and G. Yagawa, "Automatic mesh generation of quadrilateral elements using intelligent local approach", *Computer Methods in Applied Mechanics and Engineering*, vol. 179, pp. 125–138, 1999.

[66] T. S. Lau and S. H. Lo, "Finite element mesh generation over analytical curved surfaces", *Computers and Structures*, vol. 59(2), pp. 301–309, 1996.

[67] J. Rupert, "A Delaunay refinement algorithm for quality 2-dimensional mesh generation", *Journal of Algorithms*, vol. 18, pp. 548–585, 1995.

[68] K. Ho-Le, "Finite element mesh generation methods: A review and classification", *Computer Aided Design*, vol. 20(1), pp. 27–38, 1988.

[69] V. N. Kaliakin, "A simple coordinate determination scheme for two-dimensional mesh generation", *Computers and Structures*, vol. 43(3), pp. 505–516, 1992.

[70] T. K. H. Tam and C. G. Armstrong, "2D finite element mesh generation by medial axis subdivision", *Advances in Engineering Software*, vol. 18(5-6), pp. 313–324, 1991.

[71] T. K. H. Tam and C. G. Armstrong, "Finite element mesh control by integer programming", *International Journal for Numerical Methods in Engineering*, vol. 3, pp. 2581–2605, 1993.

[72] E. Ruiz-Gironés and J. Sarrate, "Generation of structured meshes in multiply connected surfaces using submapping", *Advances in Engineering Software*, vol. 41(2), pp. 379–387, 2010.

[73] G. Subramanian, A. Prasantb and V. V. S. Raveendra, "An algorithm for two- and three-dimensional automated structural mesh generation", *Computers and Structures*, vol. 61, pp. 471–477, 1996.

[74] T. Taniguchi, "An interactive automatic mesh generator for the microcomputer", *Computers and Structures*, vol. 30(3), pp. 715–722, 1988.

[75] M. S. Shephard, H. L. de Cougny, R. M. O'Bara and M. W. Beall, "Automatic grid generation using spatially based trees", in *Handbook of Grid Generation* (J. Thompson, B. Soni and N. Weatherill, eds.), ch. 15, CRC Press, 1999.

[76] K. C. Chellamuthu and N. Ida, "Algorithms and data structures for 2D and 3D adaptive finite element mesh refinement", *Finite Elements in Analysis and Design*, vol. 17, pp. 205–229, 1994.

[77] C. S. Krishnamoorthy and K. R. Umesh, "Adaptive mesh refinement for two-dimesional finite element stress analysis", *Computers and Structures*, vol. 48(1), pp. 121–133, 1993.

[78] S. H. Lo, "A new mesh generation scheme for arbitrary planar domains", *International Journal for Numerical Methods in Engineering*, vol. 21(8), pp. 1403–1426, 1985.

[79] S. H. Lo, "Generation of high-quality gradation finite element mesh", *Engineering Fracture Mechanics*, vol. 41(2), pp. 191–202, 1992.

[80] E. Hinton, N. V. R. Rao and M. Ozakca, "Mesh generation with adaptive finite element analysis", *Advances in Engineering Software*, vol. 13(5-6), pp. 238–262, 1991.

[81] J. Peraire, J. Peiro and K. Morgan, "Advancing front grid generation", in *Handbook of Grid Generation* (J. Thompson, B. Soni and N. Weatherill, eds.), ch. 17, CRC Press, 1999.

[82] R. P. Bhatia and K. L. Lawrence, "Two-dimensional finite element mesh generation based on stripwise automatic triangulation", *Computers and Structures*, vol. 36(2), pp. 309–319, 1990.

[83] A. El-Hamalawi, "A 2D combined advancing front-Delaunay mesh generation scheme", *Finite Elements in Analysis and Design*, vol. 40, pp. 967–989, 2004.

[84] X. Xu, C. C. Pain, A. J. H. Goddard and C. R. E. de Oliveira, "An automatic adaptive meshing technique for Delaunay triangulations", *Computer Methods in Applied Mechanics and Engineering*, vol. 161, pp. 297–303, 1998.

[85] T. J. Baker, "Delaunay-Voronoi methods", in *Handbook of Grid Generation* (J. Thompson, B. Soni and N. Weatherill, eds.), ch. 16, CRC Press, 1999.

[86] S. Secchi and L. Simoni, "An improved procedure for 2D unstructured Delaunay mesh generation", *Advances in Engineering Software*, vol. 34, pp. 217–234, 2003.

[87] S. H. Lo and W. X. Wang, "Generation of finite element mesh with variable size over an unbounded 2D domain", *Computational Methods in Applied Mechanics and Engineering*, vol. 194, pp. 4668–4684, 2005.

[88] K. Shimada and D. C. Gossard, "Automatic triangular mesh generation of trimmed parametric surfaces for finite element analysis", *Computer Aided Geometric Design*, vol. 15, pp. 199–222, 1998.

[89] C. K. Lee and S. H. Lo, "A new scheme for the generation of a graded quadrilateral mesh", *Computers and Structures*, vol. 5, pp. 847–857, 1994.

[90] T. D. Blacker, M. B. Stephenson and S. Canann, "Analysis automation with paving: A new quadrilateral meshing technique", *Advances in Engineering Software*, vol. 13(5/6), pp. 332–337, 1991.

[91] T. D. Blacker, "Automated quadrilateral surface discretization method and apparatus useable to generate mesh in a finite element analysis system", 1994.

[92] J. A. Shaw, "Hybrid grids", in *Handbook of Grid Generation* (J. Thompson, B. Soni and N. Weatherill, eds.), ch. 23, CRC Press, 1999.

[93] S. H. Lo and C. K. Lee, "On using meshes of mixed element types in adaptive finite element analysis", *Finite Elements in Analysis and Design*, vol. 11, pp. 307–336, 1992.

[94] Y. Kallinderis, "Hybrid grids and their applications", in *Handbook of Grid Generation* (J. Thompson, B. Soni and N. Weatherill, eds.), ch. 25, CRC Press, 1999.

[95] B. Rivière, M. F. Wheeler and V. Girault, "A priori error estimates for finite element methods based on discontinuous approximation spaces for elliptic problems", *SIAM Journal on Numerical Analysis*, vol. 39(3), pp. 902–931, 2001.

[96] T. Grätsch and K.-J. Bathe, "A posteriori error estimation techniques in practical nite element analysis", *Computers and Structures*, vol. 83, pp. 235–265, 2005.

[97] Y. Luo, "Adaptive nearest-nodes finite element method guided by gradient of linear strain energy density", *Finite Elements in Analysis and Design*, vol. 45, pp. 925–933, 2009.

[98] O. C. Zienkiewicz and J. Z. Zhu, "The superconvergent patch recovery and a posteriori error estimators. part 1. the recovery technique", *International Journal for Numerical Methods in Engineering*, vol. 33, pp. 1331–1364, 1992.

[99] O. C. Zienkiewicz and J. Z. Zhu, "The superconvergent patch recovery and a posteriori error estimators. part 2. error estimates and adaptivity", *International Journal for Numerical Methods in Engineering*, vol. 33, pp. 1365–1382, 1992.

[100] C. Erath, S. Ferraz-Leite, S. Funken and D. Praetorius, "Energy norm based a posteriori error estimation for boundary element methods in two dimensions", *Applied Numerical Mathematics*, vol. 59(11), pp. 2713–2734, 2009.

[101] J. F. Shepherd, M. W. Dewey, A. C.Woodbury, S. E. Benzley, M. L. Staten and S. J. Owen, "Adaptive mesh coarsening for quadrilateral and hexahedral meshes", *Finite Elements in Analysis and Design*, vol. 46, pp. 17–32, 2010.

[102] G. H. Wang, J. M. Tyler, J. S. Weltman and J. D. Callahan, "Node-based dynamic adaptive grid with quadrilateral and hexahedral elements", *Advances in Engineering Software*, vol. 30, pp. 31–41, 1999.

[103] R. Aubry, G. Houzeaux and M. Vázquez, "A surface remeshing approach", *International Journal for Numerical Methods in Engineering*, vol. 85, pp. 1475–1498, 2011.

[104] T. C. S. Rendall and C. B. Allen, "Reduced surface point selection options for efficient mesh deformation using radial basis functions", *Journal of Computational Physics*, vol. 229, pp. 2810–2820, 2010.

[105] S. H. Lo, "Finite element mesh generation over curved surfaces", *Computers and Structures*, vol. 29(5), pp. 732–742, 1988.

[106] S. H. Lo, "Generating quadrilateral elements on plane and over curved surfaces", *Computers and Structures*, vol. 31(3), pp. 421–426, 1989.

[107] J. Peiro, "Surface grid generation", in *Handbook of Grid Generation* (J. Thompson, B. Soni and N. Weatherill, eds.), ch. 19, CRC Press, 1999.

[108] T. S. Lau, S. H. Lo and C. K. Lee, "Generation of quadrilateral mesh over analytical curved surfaces", *Finite Elements in Analysis and Design*, vol. 27, pp. 251–272, 1997.

[109] C. C. L. Wang and K. Tang, "Non-self-overlapping Hermite interpolation mapping: a practical solution for structured quadrilateral meshing", *Computer-Aided Design*, vol. 37, pp. 271–283, 2005.

[110] A. Gelas, S. Valette, R. Prost and W. L. Nowinski, "Variational implicit surface meshing", *Computers and Graphics*, vol. 33, pp. 312–320, 2009.

[111] P. Cignoni, C. Montani and R. Scopigno, "A comparison of mesh simplification algorithms", *Computers and Graphics*, vol. 22(1), pp. 37–54, 1998.

[112] J.-D. Boissonnat and S. Oudot, "Provably good sampling and meshing of surfaces", *Graphical Models*, vol. 67, pp. 405–451, 2005.

[113] A. C. O. Miranda, L. F. Martha, P. A. Wawrzynek and A. R. Ingraffea, "Surface mesh regeneration considering curvatures", *Engineering with Computers*, vol. 25, pp. 207–291, 2009.

[114] J.-F. Remacle, C. Geuzaine, G. Compère and E. Marchandise, "High-quality surface remeshing using harmonic maps", *International Journal for Numerical Methods in Engineering*, vol. 83, pp. 403–425, 2010.

[115] P. Alliez, G. Ucelli, C. Gotsman and M. Attene, "Recent advances in remeshing of surfaces", in *Shape Analysis and Structuring* (L. Floriani and M. Spagnuolo, eds.), pp. 53–82, Springer, 2008.

[116] M. Alexa, "Recent advances in mesh morphing", *Computer Graphics Forum*, vol. 21(2), pp. 173–197, 2002.

[117] J. F. Thompson, B. K. Soni and N. P. Weatherill, *Handbook of Grid Generation*. CRC Press, 1999.

[118] B. M. Irons and O. C. Zienkiewicz, "The isoparametric finite element system - a new concept in finite element analysis", in *Recent Advances in Stress Analysis, Royal Aeronautical Society, London, UK*, 1968.

[119] J. C. Lachat and J. O. Watson, "Effective numerical treatment of boundary integral equations: A formulation for three-dimensional elastostatics", *International Journal for Numerical Methods in Engineering*, vol. 10, pp. 991–1005, 1976.

[120] G. G. W. Mustoe, "Advanced integration schemes over boundary elements and volume cells for two- and three-dimensional non-linear analysis", in *Developments in Boundary Element Methods - 3* (P. Banerjee and S. Mukharjee, eds.), pp. 213–270, Elsevier, London, 1984.

[121] X. W. Gao and T. G. Davies, "Adaptive integration in elasto-plastic boundary element analysis", *Journal of the Chinese Institute of Engineers*, vol. 23(3), pp. 349–356, 2000.

[122] S. Bu and T. G. Davies, "Effective evaluation of non-singular integrals in 3D BEM", *Advances in Engineering Software*, vol. 23, pp. 121–128, 1995.

[123] L. Roklin and V. Greengard, "A fast algorithm for particle simulations", *Journal of Computational Physics*, vol. 73, pp. 325–348, 1987.

[124] W. Hackbusch and Z. P. Nowak, "On the fast matrix multiplication in the boundary element method by panel clustering", *Numerische Mathematik*, vol. 54, pp. 463–491, 1989.

[125] W. Hackbusch, "A sparse matrix arithmetic based on H-matrices. part I: Introduction to H-matrices", *Computing*, vol. 62, pp. 89–108, 1999.

[126] M. Bebendorf, "Approximation of boundary element matrices", *Numerische Mathematik*, vol. 86, pp. 565–589, 2000.

[127] S. Börm, L. Grasedyck and W. Hackbusch, "Introduction to hierarchical matrices with applications", *Engineering Analysis with Boundary Elements*, vol. 27, pp. 405–422, 2003.

[128] T. M. Foster, M. S. Mohamed, J. Trevelyan and G. Coates, "Rapid re-meshing and re-solution of three-dimensional boundary element problems for interactive stress analysis", *Engineering Analysis with Boundary Elements*, vol. 36, pp. 1331–1343, 2012.

[129] M. Bebendorf and S. Rjasanow, "Adaptive low-rank approximation of collocation matrices", *Computing*, vol. 70, pp. 1–24, 2003.

[130] M. Bebendorf and R. Kriemann, "Fast parallel solution of boundary integral equations and related problems", *Computing and Visualization in Science*, vol. 8, pp. 121–135, 2005.

[131] S. Kurz, "The adaptive cross-approximation technique for the 3D boundary-element method", *IEEE Transactions on Magnetics*, vol. 38(2), pp. 421–424, 2002.

[132] W. Weber, K. Kolk and G. Kuhn, "Acceleration of 3D crack propagation simulation by the utilization of fast BEM-techniques", *Engineering Analysis with Boundary Elements*, vol. 33, pp. 1005–1015, 2009.

[133] K. Frederix and M. van Barel, "Solving a large dense linear system by adaptive cross approximation", *Journal of Computational and Applied Mathematics*, vol. 234, pp. 3181–3195, 2010.

[134] F. Maerten, "Adaptive cross-approximation applied to the solution of system of equations and post-processing for 3D elastostatic problems using the boundary element method", *Engineering Analysis with Boundary Elements*, vol. 34, pp. 483–491, 2010.

[135] V. Mallardo, M. H. Aliabadi, A. Brancati and V. Marant, "An accelerated BEM for simulation of noise control in the aircraft cabin", *Aerospace Science and Technology*, vol. 23(1), pp. 418–428, 2012.

[136] J. Trevelyan and D. J. Scales, "Rapid re-analysis in BEM elastostatic calculations", in *28th International Conference on Boundary Element / Mesh Reduction Methods, Greece, USA*, 2006.

[137] S. Cotin, H. Delingette and N. Ayache, "Real-time elastic deformations of soft tissues for surgery simulation", *IEEE Transactions on Visualization and Computer Graphics*, vol. 5(1), pp. 62–73, 1999.

[138] U. Kirsch and G. Toledano, "Approximate reanalysis for modifications of structural geometry", *Computers and Structures*, vol. 16(1), pp. 269–277, 1983.

[139] H. van der Vorst, "Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems", *SIAM Journal of Scientific and Statistical Computing*, vol. 13, pp. 631–644, 1992.

[140] R. W. Freund, "A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems", *SIAM Journal on Scientific Computing*, vol. 14(2), pp. 470–482, 1993.

[141] M. García, O. D. Robles, L. Pastor and A. Rodríguez, "MSRS: A fast linear solver for the real-time simulation of deformable objects", *Computers and Graphics*, vol. 32, pp. 293–306, 2008.

[142] M. Kauers, "Fast solvers for dense linear systems", *Nuclear Physics B (Proceedings Supplements)*, vol. 183, pp. 245–250, 2008.

[143] L. P. S. Barra, A. L. G. A. Coutinho, W. J. Mansur and J. C. F. Telles, "Iterative solution of BEM equations by GMRES algorithm", *Computers and Structures*, vol. 44(6), pp. 1249–1253, 1992.

[144] W. Y. Kong, J. Bremer and V. Rokhlin, "An adaptive fast direct solver for boundary integral equations in two dimensions", *Applied and Computational Harmonic Analysis*, vol. 31(3), pp. 346–369, 2011.

[145] J. Trevelyan and P. Wang, "Interactive re-analysis in mechanical design evolution. part I. background and implementation", *Computers and Structures*, vol. 79, pp. 929–938, 2001.

[146] J. Trevelyan and P. Wang, "Interactive re-analysis in mechanical design evolution. part II. rapid evaluation of boundary element integrals", *Computers and Structures*, vol. 79, pp. 939–951, 2001.

[147] J. Trevelyan, D. J. Scales and S. H. Spence, "Interactive display of stress contours in real time", in *NAFEMS World Congress, Vancouver, Canada*, 2007.

[148] E. Kreyszig, *Advanced Engineering Mathematics (8th edition)*. Wiley, 1999.

[149] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. M. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, 1994.

[150] N. Nachtigal, S. Reddy and L. Trefethen, "How fast are nonsymmetric matrix iterations?", *SIAM Journal on Matrix Analysis and Applications*, vol. 13, pp. 778–195, 1992.

[151] G. A. Gravvanis and K. M. Giannoutakis, "Fast parallel finite element approximate inverses", *Computational Modelling in Engineering and Sciences*, vol. 32(1), pp. 35–44, 2008.

[152] H. Courtecuisse, J. Allard, C. Duriez and S. Cotin, "Asynchronous preconditioners for efficient solving of non-linear deformations", in *7th Workshop on Virtual Reality Interaction and Physical Simulation, Copenhagen, Denmark*, 2010.

[153] R. Freund and N. Nachtigal, "QMR: A quasi-minimal residual method for non-Hermitian linear systems", *Numerische Mathematik*, vol. 60, pp. 315–339, 1991.

[154] U. Kirsch, "Reduced basis approximations of structural displacements for optimal design", *American Institute of Aeronautics and Astronautics Journal*, vol. 29, pp. 1751–1758, 1991.

[155] M. Xiao, P. Breitkopf, R. Coelho, C. Knopf-Lenoir and P. Villon, "Enhanced POD projection basis with application to shape optimization of car engine intake port", *Structural and Multidisciplinary Optimization*, vol. 46, pp. 129–136, 2012.

[156] D. Ryckelynck, "A priori hyperreduction method: an adaptive approach", *Journal of Computational Physics*, vol. 202, pp. 246–266, 2005.

[157] M. A. Woodbury, "Inverting modified matrices", *Memorandum Rept. 42, Statistical Research Group, Princeton University, Princeton, NJ*, 1950.

[158] J. Sherman and W. Morrison, "Adjustment of an inverse matrix corresponding to a change in one element of a given matrix", *Annals of Mathematical Statistics*, vol. 21(1), pp. 124–127, 1950.

[159] B. M. Irons, "A frontal solution scheme for finite element analysis", *International Journal for Numerical Methods in Engineering*, vol. 2, pp. 5–32, 1970.

[160] J. A. Scott, "Parallel frontal solvers for large sparse linear systems", *ACM Transactions on Mathematical Software*, vol. 29(4), pp. 395–417, 2003.

[161] V. Eijkhout, "Overview of iterative linear system solver packages", *NHSE Review*, vol. 3, 1998.

[162] Nvidia, *CUDA Toolkit 4.0 CUBLAS Library*, 2011.

[163] R. Couturier and S. Domas, "Sparse systems solving on GPUs with GMRES", *Journal of Supercomputing*, vol. 59(3), pp. 1504–1516, 2012.

[164] N. Galoppo, N. K. Govindaraju, M. Henson and D. Manocha, "LU-GPU: Efficient algorithms for solving dense linear systems on graphics hardware", in *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, Seattle, USA*, 2005.

[165] C. Geuzaine and J.-F. Remacle, "Gmsh: A 3D finite element mesh generator with built-in pre- and post-processing facilities", *International Journal for Numerical Methods in Engineering*, vol. 79, pp. 1309–1331, 2009.

[166] "Open Cascade." www.opencascade.org, 2013.

[167] R. Löhner, J. Camberos and M. Merriam, "Parallel unstructured grid generation", *Computer Methods in Applied Mechanics and Engineering*, vol. 95, pp. 343–357, 1992.

[168] I. M. Smith, L. Margetts, G. Beer and C. Duenser, "Parallelising the boundary element method using ParaFEM", in *Tenth International Symposium on Numerical Models in Geomechanics, Rhodes, Greece, Balkema, Netherlands*, 2007.

[169] I. M. Smith and L. Margetts, "The convergence variability of parallel iterative solvers", *Engineering Computations*, vol. 23(2), pp. 154–165, 2006.

[170] W. D. Pilkey and D. F. Pilkey, *Peterson's Stress Concentration Factors (Third Edition)*. John Wiley & Sons, 2008.

[171] T. J. R. Hughes, J. A. Cottrell and Y. Bazilevs, "Isogeometric analysis: CAD, finite elements, NURBS, exact geometry, and mesh refinement", *Computer Methods in Applied Mechanics and Engineering*, vol. 194, pp. 4135–4195, 2005.

[172] R. N. Simpson, S. P. A. Bordas, J. Trevelyan and T. Rabczuk, "A two-dimensional isogeometric boundary element method for elastostatic analysis", *Computational Methods in Applied Mechanics and Engineering*, vol. 209-212, pp. 87–100, 2012.

[173] M. A. Scott, R. N. Simpson, J. A. Evans, S. Liptona, S. P. A. Bordas, T. J. R. Hughes and T. W. Sederberg, "Isogeometric boundary element analysis using unstructured t-splines", *Computer Methods in Applied Mechanics and Engineering*, vol. 254, pp. 197–221, 2013.

Glossary

Advancing front	A meshing procedure whereby elements are generated starting from the boundary and working inwards.
Circumcircle	A circle that passes through all three vertices of a triangle. The centre of the circumcircle is called the circumcentre.
Delaunay triangulation	A meshing procedure whereby nodes spread across a surface are connected by a series of triangles. To be classed as a Delaunay mesh, the circumcircle of each triangle must not contain any corner nodes of other triangles.
Edge	A line segment joining two vertices in a polygon.
Element	A shape defining part of a discretised surface in two or three dimensions.
Element quad	A group of four elements around a single node.
Element triplet	A group of three elements around a single node.
Face	A two- or three-dimensional polygon which makes up part of the surface of a polyhedron. A face is always bounded by edges.
Field element	The element which is paired with a source point to carry out integration.
Field point	A point on a surface which is used to construct the integral over the surface.
Gauss point	A point at a specific location on an element which is used in numerical integration.
Haptic feedback	Real-time physical feedback, typically provided by means of hand held device, that gives the feel of interaction with an object.
Incircle	The largest circle that fits within a triangle. It touches and is tangent to the three sides. The centre of the incircle is called the triangle's incentre.
Interactive	A process provides feedback within a few seconds.
Laplacian smoothing	Moving a node to the average coordinates of all the nodes to which it is connected in a mesh.
Line	A geometric entity in three- (x, y, z) or two-dimensional (u, v) space. A line defines part of or the entire boundary of a patch. It is defined by a vertex at each end and additional data relating to its path.
Near-singular integral	Occurs as $r \to 0$ but $r \neq 0$ where r is the denominator of an equation.

Node	A point at which stresses and displacements are calculated. Each element has several nodes associated with it.
Octree	A meshing technique whereby cubic elements are successively divided into eight to refine the mesh.
Orthogonal	The objects are perpendicular to each other.
Patch	A two- or three-dimensional surface bounded by a set of lines. A patch will always lie on a face but may not necessarily cover the entire area.
Quadtree	A meshing technique whereby square elements are successively divided into four to refine the mesh.
Rapid	The process takes place in the time a user is prepared to wait (typically a few seconds).
Real-time	A process takes place with a higher frequency refresh rate than a human can detect.
Re-analysis	The process of re-integrating and re-solving an updated boundary element mesh.
Re-entry corner	An internal angle between two consecutive lines bounding a patch of greater than π radians.
Re-generation	Reconstructing a mesh over all or a specific area of a model.
Re-integration	Updating an existing BE system of equations due to a geometric change.
Re-meshing	Updating an existing mesh based on a geometric change.
Re-solve	Finding the new solution to an updated linear system of equations.
Segment	A line forming a single side of an element.
Shape function	A mathematical formula used to interpolate across an element.
Singular integral	Occurs when the denominator of an equation approaches zero. When $r \to 0$, $1/r$ is weakly singular and $1/r^2$ is strongly singular.
Source point	A point on a surface at which a fictitious load is applied so that integration can be carried out over the field points or elements.
System architecture	The bandwidth of the CPU in a computer. Typically 32- or 64-bits.
System matrix	Matrix $[A]$ in the linear equation $[A]\{x\} = \{b\}$. This matrix contains the integrals of the traction and displacement kernels.
Vertex	A point defining a 'corner' of a model or patch, or the end of a line in three- (x, y, z) or two-dimensional (u, v) space.
Wall clock time	The amount of time taken by a process measured by the CPU clock.
Wire	A complete loop of lines which form the boundary of, or an internal feature within, a patch.

Appendix A

Solver pseudo-code

The iterative solver pseudo-code for the generalised minimal residual (GMRES), bi-conjugate gradient stabilised (BiCGSTAB) and transpose free quasi-minimal residual (TFQMR) schemes, presented in Algorithms A.1, A.2 and A.3, is based on the templates given in [149] with some modifications made by the author to improve efficiency. Leu's reduction, given in Algorithm A.4, eigenvector based proper orthogonal decomposition (E-POD) given in Algorithm A.5 and singular value decomposition based proper orthogonal decomposition (SVD-POD) given in Algorithm A.6, together with the proper orthogonal decomposition (POD) solution scheme given in Algorithm A.7 are based on algorithms found in [23], [25] and [26] respectively. The nomenclature used in all the solvers is given in in Table A.1 and the solver specific nomenclature in Table A.2. Note that this may differ from the standard notation as some vectors have been reused to reduce the memory size and access requirements of the solvers.

Table A.1: General variables.

Symbol	**Description**
A	new system matrix
x_0, x	previous and current solution vectors
b	new b vector
r_0, r	initial and current residual vectors
M	preconditioning matrix
$\hat{n}$	limit on number of solver iterations
tol	solution tolerance
$\lvert \cdot \rvert$	L_1-norm
$\lVert \cdot \rVert$	L_2-norm

Table A.2: Solver specific variables.

Algorithm	**Variables**	**Vectors**	**Matrices**
GMRES[1]	α, θ, τ	c, g, s, u, v, w, y	H, V
BiCGSTAB	$\alpha, \beta, \rho_1, \rho_2, \tau, \omega$	p, s, t, v	
TFQMR	$\alpha, \beta, \gamma, \eta, \theta, \rho_1, \rho_2, \tau$	d, g, h, u, v, w	
Leu's reduction		$\alpha, \beta, \zeta, \varphi, \psi$	ΔA
POD[2]		u, v, λ, ζ	Φ, K, Q, V

[1]GMRES matrix V contains vectors v.
[2]POD matrix Q contains vectors x, matrix U contains vectors u and matrix V contains vectors v

Algorithm A.1 GMRES

$r_0 = b - Ax_0$
$\tau = 1/\|b\|$
$g_0 = \|r_0\|$
$v_0 = r_0/g_0$
$i = 0$
while $i < \hat{n}$ **do**
    Solve: $Mw = Av_i$
    for $j = 0, 1, 2, ..., i$ **do**
        $H_{i,j} = w^T v_j$
        $w = w - H_{i,j} v_j$
    end for
    $v_{i+1} = w/\|w\|$
    for $j = 0, 1, 2, ..., i$ **do**
        $\alpha = H_{i,j} c_j - H_{i,j+1} s_j$
        $H_{i,j+1} = H_{i,j} s_j - H_{i,j+1} c_j$
        $H_{i,j} = \alpha$
    end for
    $\theta = -\tan^{-1}(\|w\|/H_{i,i})$
    $c_i = \cos\theta$
    $s_i = \sin\theta$
    $H_{i,i} = H_{i,i} c_i - \|w\| s_i$
    $g_{i+1} = g_i s_i$
    $g_i = g_i c_i$
    if $\text{abs}(g_{i+1})\tau < tol$ **then**
        Converged: end while
    end if
    $i = i + 1$
end while
Solve: $H^T y = g$
$u = V^T y$
Solve: $Mw = u$
$x = x_0 + w$

Algorithm A.2 BiCGSTAB

$r_0 = b - Ax_0$
$\tau = 1/\|b\|$
$r = r_0$
$x = x_0$
$i = 0$
while $i < \hat{n}$ **do**
    $\rho_1 = r_0^T r$
    if $i = 0$ **then**
        $p = r$
    else
        $\beta = (\rho_1/\rho_2)(\alpha/\omega)$
        $p = r + \beta(p - \omega v)$
    end if
    Solve: $Ms = p$
    $v = As$
    $\alpha = \rho_1/r_0^T v$
    $x = x + \alpha s$
    $r = r - \alpha v$
    if $\|r\|\tau < tol$ **then**
        Converged: end while
    end if
    Solve: $Ms = r$
    $t = As$
    $\omega = t^T r/t^T t$
    $x = x + \omega s$
    $r = r - \omega t$
    if $\|r\|\tau < tol$ **then**
        Converged: end while
    end if
    $\rho_2 = \rho_1$
    $i = i + 1$
end while

Algorithm A.3 TFQMR

Solve: $Mr = b - Ax_0$
Solve: $Mv = Ar$
$g = v$
$w = u = r$
$\rho_1 = r^T r$
$\tau = \sqrt{\rho_1}$
$x = x_0$
$i = 0$
while $i < \hat{n}$ **do**
    $\alpha = \rho_1 / r^T v$
    if $i = 0$ **then**
        $d = u$
    else
        $\beta = \theta^2 \eta / \alpha$
        $d = u + \beta d$
    end if
    $w = w - \alpha g$
    $\theta = \|w\| / \tau$
    $\gamma = 1/(1 + \theta^2)$
    $\eta = \gamma \alpha$
    $\tau = \tau \theta \sqrt{\gamma}$
    $x = x + \eta d$
    $i = i + 1$
    if $\tau\sqrt{i} < tol$ **then**
        Converged: end while
    end if
    $u = u - \alpha v$
    Solve: $Mh = Au$
    $\beta = \theta^2 \gamma$
    $d = u + \beta d$
    $w = w - \alpha h$
    $\theta = \|w\| / \tau$
    $\gamma = 1/(1 + \theta^2)$
    $\eta = \gamma \alpha$
    $\tau = \tau \theta \sqrt{\gamma}$
    $x = x + \eta d$
    $i = i + 1$
    if $\tau\sqrt{i} < tol$ **then**
        Converged: end while
    end if
    $\rho_2 = r^T w$
    $\alpha = \rho_2 / \rho_1$
    $u = w + \alpha u$
    Solve: $Mg = Au$
    $v = g + \alpha(h + \alpha v)$
    $\rho_1 = \rho_2$
end while

Algorithm A.4 Leu's reduction

Solve: $A_0 \varphi_0 = b$
$\varphi_0 = \varphi_0 / |\varphi_0|$
$\psi_0 = \varphi_0 / (\varphi_0^T A \varphi_0)$
$\zeta_0 = \psi_0^T b$
$x = \zeta_0 \varphi_0$
$\Delta A = A - A_0$
$i = 1$
while $i < \hat{n}$ **do**
    Solve: $A_0 \varphi_i = \Delta A \varphi_{i-1}$
    $\varphi_i = \varphi_i / |\varphi_i|$
    $\alpha = A \varphi_i$
    $\beta = 0$
    for $j = 0, 1, 2, ..., i - 1$ **do**
        $\beta = \beta + \psi_j^T \alpha \varphi_j$
    end for
    $\varphi_i = \varphi_i - \beta$
    $\psi_i = \varphi_i = \varphi_i / |\varphi_i|$
    $\alpha = \psi_i^T A$
    $\beta = 0$
    for $j = 0, 1, 2, ..., i - 1$ **do**
        $\beta = \beta + \alpha \varphi_j \psi_j$
    end for
    $\psi_i = \psi_i - \beta$
    $\psi_i = \psi_i / (\psi_i^T A \varphi_i)$
    $\zeta_i = \psi_i^T b$
    $x = x + \zeta_i \varphi_i$
    if $\text{abs}(\zeta_i / |\zeta|) < tol$ **then**
        Converged: end while
    end if
    $i = i + 1$
end while

Algorithm A.5 E-POD construction

1: $Q = \begin{bmatrix} x_0 & x_1 & \dots & x_s \end{bmatrix}$
2: $K = QQ^T$
3: Solve eigenproblem: $VK = \lambda K$
4: Order m columns of V where $\lambda_j > 10^{-10}\lambda_{\max}$
5: $\Psi = \begin{bmatrix} v_0 & v_1 & \dots & v_m \end{bmatrix}$

Algorithm A.6 SVD-POD construction

1: $Q = \begin{bmatrix} x_0 & x_1 & \dots & x_s \end{bmatrix}$
2: Find SVD: $Q = USV^T$
3: if $U = \begin{bmatrix} u_0 & u_1 & \dots & u_s \end{bmatrix}$, $\Psi = \begin{bmatrix} u_0 & u_1 & \dots & u_m \end{bmatrix}$, where $m < s$

Algorithm A.7 POD solution

1: $\hat{A} = \Psi^T A_i \Psi$
2: $\hat{b} = \Psi^T b_i$
3: Solve: $\hat{A}\zeta = \hat{b}$
4: $x_i = \Psi\zeta$