a

b

Fig. 2.7 Node numbering for the 8- to 26-node linear, isoparametric hexahedron
 a. Vertex nodes and optional midedge nodes; b. Optional midface nodes

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act; R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

**An Adaptive Mesh-Management Algorithm for
Three-Dimensional Finite Element Analysis**

by

Jennifer Moyra Jeane McDill
B. Eng. (Mechanical), M. Eng. (Materials)

A thesis submitted to the Faculty
of Graduate Studies And Research
in partial fulfilment of the
requirements for the degree of
Doctor of Philosophy in Engineering

Department of
~~Mechanical and Aeronautical Engineering~~
Carleton University
Ottawa, Ontario
Canada
March 1988

©1988
Jennifer Moyra Jeane McDill

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-40611-9

An Adaptive Mesh-Management Algorithm for Three-Dimensional Finite Element Analysis

J. M. J. McDill, B.Eng., M.Eng.
in partial fulfilment of the requirements
for the degree of Doctor of Philosophy

Chairman, Department of
Mechanical and Aeronautical Engineering

R. Canessa

Carleton University
March 1988

Abstract

Many finite element problems with localized phenomena are solved more efficiently with a fine discretization in one area and a coarser discretization elsewhere. An analysis with a uniformly fine mesh is often prohibitively expensive. In many time-dependent problems it is desirable for the region of fine discretization to move about the domain. Mesh-grading elements and dynamic mesh management can substantially reduce the costs of analysis for problems of this type.

Two-dimensional (2-D) and three-dimensional (3-D) mesh-grading finite elements for problems with localized phenomena are presented. These quadrilaterals and hexahedrons permit mesh grading without element distortion and, if desired, can be used in a convenient recursive form. Constraints associated with mesh grading are embedded in the basis functions, making the elements computationally efficient and easily implemented in standard finite element programs. Dramatic computational savings have been achieved. The computational complexity for an implicit analysis of a point load on a 3-D uniform mesh is $O(n^7)$. For the equivalent recursively graded mesh the computational complexity is $O(\log_2 n)$. The corresponding 2-D analyses have computational complexities of $O(n^4)$ and $O(\log_2 n)$. The 8- to 26-node hexahedron was instrumental in enabling the first fully 3-D thermal-elasto-plastic analysis of a weld. This advance in mesh grading has allowed a workstation to perform 3-D thermal-elasto-plastic analysis faster than supercomputers using traditional analysis.

The mesh-grading elements allow local changes to be conveniently made in finite element meshes. As such, they are naturally suited to adaptive mesh-management algorithms. A rule-based mesh-management algorithm which exploits both 3-D mesh-grading elements and spatially addressable data structures is presented. This algorithm automatically refines and coarsens a finite element mesh in response to element level error indicators. The resulting finite element meshes are hierarchical and dynamic. The adaptive mesh-management algorithm, when used in combination with other research, will provide an essential step towards fully automatic and adaptive finite element analysis.

Acknowledgements

It is a distinct pleasure for me to have reached the stage at which I am able to express, formally, my gratitude to those who have helped me over the last four years. My advisor, Dr John Goldak, has been my mentor for many years and has given me a tremendous amount of sound advice. In retrospect, it has been Dr Goldak's dynamic personality that has inspired me in much of my work. Others, particularly Dr Neil Holtz, Dan Downey, Marc Paquin, Stephanie Hunn, Warren Hik and Balvant Patel have given me invaluable technical assistance. I am also deeply grateful for the financial support of Carleton University, the Natural Sciences and Engineering Research Council (NSERC) and the Ontario Graduate Studies (OGS) program. Finally, my parents have been a constant source of encouragement and support; my husband, Alan, has shared the dream and the long hours of study.

Sic itur ad astra.

Table of Contents

1.	Introduction	1
2.	Isoparametric Elements for Mesh-Management Algorithms	4
2.1	<i>Mesh-Grading Elements</i>	6
2.2	<i>Mesh Grading in Two Dimensions</i>	11
2.3	<i>Mesh Grading in Three Dimensions</i>	24
2.4	<i>Computational Effectiveness of Mesh-Grading Elements</i>	33
3.	An Adaptive Mesh-Management Algorithm	40
3.1	<i>Dynamic Mesh Management</i>	42
3.2	<i>Formulation</i>	63
3.3	<i>Refinement</i>	64
3.4	<i>Coarsening</i>	69
3.5	<i>Validation of the Mesh-Management Algorithm</i>	88
4.	Automatic and Adaptive Finite Element Analysis	94
4.1	<i>Data Transfer</i>	94
4.2	<i>Error Estimates</i>	96
4.3	<i>Advances for Error Indicators</i>	98
4.4	<i>Large Deformation</i>	99
4.5	<i>Fully Automatic Finite Element Analysis</i>	101
5.	Conclusions and Contributions	102
6.	References	103
	Appendix I	113
	Appendix II	115
	Appendix III	123
	Appendix IV	128
	Appendix V	135

List of Figures

2.1	Mesher for 2-D analysis	5
2.2	A graded mesh showing irregular nodes as open circles	8
2.3	Numbering for the 4- to 8-node linear, isoparametric quadrilateral	12
2.4	4- to 8-node linear quadrilateral with one optional node	14
2.5	Mesh for comparison of element types	19
2.6	Projected Stresses σ_{yy}	20
2.7	Node numbering for the 8- to 26-node linear, isoparametric hexahedron	25
2.8	8- to 26-node hexahedron with 2 optional nodes	26
2.9	Graded 3-D mesh with 3 levels of recursion	31
2.10	Temperature vs radius for the $8 \times 8 \times 8$ uniform mesh and graded mesh with 3 levels of recursion	32
2.11	Displacement field for mesh-grading elements used in computational weld mechanics	34
2.12	Graded meshes with the same global stiffness matrix	35
2.13	Temperature field for a butt weld	38
2.14	Displacement field for a butt weld	39
3.1	Refinement in two dimensions	59
3.2	Propagation of refinement from a small element to a large element	60
3.3	Propagation of refinement from a large element to a small element	61
3.4	Structure of the adaptive mesh-management algorithm	84
3.5	Refinement and propagation	90
3.6	Refinement and coarsening	91
3.7	A moving singularity	93

List of Tables

2.1	Basis Functions for the 2-D Grading Element	13
2.2	Condition Numbers for Quadrilaterals	16
2.3	Elemental Contribution to Relative Errors, ϵ_e	21
2.4	Basis Functions for the 3-D Grading Element	27
2.5	Cost of Factoring the Stiffness Matrix	36

Nomenclature

A'	an array containing the topology of the mesh
a, b, c	integration limits
B	nodal reactions
C'	an array containing the nodal co-ordinates
C^0	the function is continuous, its first derivative is not
C^1	the function and its first derivative are continuous
C^∞	the function and all its derivatives are continuous
C_f	cost of factoring (reduction and back-substitution) the stiffness matrix
CN	the condition number of a matrix
d	integration limit
d_f	number of degrees of freedom in the problem
D'	an array containing the number of nodes in each element
e	integration limit
$\tilde{e}^i$	an element
$\tilde{e}^m$	a neighbour of element $\tilde{e}^i$
$\tilde{e}_h$	a subelement of $\tilde{e}^i$
E_0	total strain energy in the domain
f	integration limit
$file$	an input or output file
F	a factor which controls the range of elements to be coarsened
f_w	frontwidth of the problem
G'	an array containing the elements joined by each vertex node
h	summation counter or subscript
H'	an array containing the elements joined by each midedge node
i	summation counter or subscript
$info$	data extracted from or entered in a k-D tree
I'	an array containing the elements joined by each midsurface node
j	co-ordinate subscript associated with node j
k	thermal conductivity
key	the key used to extract data from or enter data in a k-D tree
K	stiffness matrix
l	summation counter or subscript
L	the length of a bar to be analysed
m	summation counter or subscript
n	number of elements on an edge of a uniform mesh
n_v	a vertex node
n_e	a midedge node
n_s	a midsurface node

N	the number of elements in the domain
N_c	the set of vertex nodes of an element
N_e	the set of midedge nodes of an element
N_j, N_k	basis functions for a new mesh
N_s	the set of midsurface nodes of an element
$\bar{N}_i$	basis functions for an old mesh
p	degree of a polynomial
P_i	basis function for node i
P_i^L	basis function for node i based on a 4-node linear quadrilateral
Q	point thermal load
r	number of recursions in a graded mesh
R	radial distance from the point thermal load
$\mathcal{R}$	the root mesh
SF	the number of significant figures
t_c	the k-D tree for finding nodal identities from co-ordinates
t_p	the k-D tree for finding parent topology from co-ordinates
T	temperature at a point
T_i	temperature at node i
T_j, T_k	nodal temperatures for a new mesh
$\bar{T}_i$	nodal temperatures for an old mesh
T_∞	ambient temperature
v_k	a column vector
V	the number of nodes in a particular element
W, W'	Gauss weights for a sampling point in Gaussian integration
X	a constrained node
Γ	a tolerance in the solution prescribed by the analyst
δ	Kronecker delta
ϵ_e	elemental contribution to the relative error
ϵ_n	global relative error
ζ, ζ'	co-ordinates in the computational domain
η, η'	co-ordinates in the computational domain
Θ_e	elemental contribution to the error in the strain energy
λ_{max}	the maximum eigenvalue of a stiffness matrix
λ_{min}	the minimum eigenvalue of a stiffness matrix
$\xi; \xi'$	co-ordinates in the computational domain
σ_{yy}	stress for y direction
Ω	the finite element domain

1. Introduction

The computer has been a tool of engineering design and engineering analysis for many years. Researchers have developed special approaches and software to help engineers use this tool. These methods are often called computer aided design and computer aided engineering. Geometric modelling and general purpose finite element analysis codes are two areas which have received much attention. Significant improvements in engineering productivity have been realized.^[1] Even so, every step of the process usually involves engineers. While the engineer should guide the process, rigorous computational methods can replace much of the time-consuming and very tedious labour performed by engineers.

Finite element analysis (FEA) provides a powerful tool which can assist designers and analysts in determining how a part, for example, responds to a given set of loads. Finite element meshes are typically generated by users, with or without mesh generators, and are guided by the heuristic experience of the user. Many rules-of-thumb for mesh generation can be found in the literature and in industry. These methods help the user to obtain a better mesh. Normally, the best meshes are as fine as can be afforded, with elements which have minimal initial distortion.

Generating the mesh topology and associated data such as boundary conditions consumes valuable time. It is well known that 60 to 90 percent of the cost of a finite element analysis is associated with preprocessing and postprocessing. Even so, there is no guarantee that the resulting meshes are optimal, or nearly optimal, for the problems considered. Relying on the experience of the user means that long training periods are required. Furthermore, some finite element analyses are outside the abilities and qualifications of many analysts who are using, or would benefit from using, these methods. Additional problems occur when inexperienced analysts use inappropriate finite element techniques. A common problem for inexperienced analysts is using a small deformation formulation on a problem which, in fact, requires a large deformation formulation.

Serious researchers are predicting that engineering workstations will have the power of current supercomputers in the early 1990's. Therefore, the pacing research issues are not those of developing new hardware, but those relating to the development of software which will automate finite element methods. Automatic finite element methods will generate near

optimal meshes and perform analyses which use the correct mathematics and physics.

Automatic methods are being studied by many researchers such as Kela^{[2][3][4]} and Shepard^{[1][5]}. For example, a finite element mesh can be generated from a solids modeller, allowing the user to simply describe the part without having to choose the specific discretization. The mesh evolves in response to some form of error measure, so a near optimal mesh is available.

Automatic FEA will extend the use of finite element analysis to designers who are not experts in numerical analysis. The goal of this research has been the development of an adaptive mesh-management algorithm for automatic FEA. In combination with other research at Carleton, this algorithm will contribute to the development of fully automatic finite element analysis of manufacturing processes such as welding.

It is widely recognized that finite element analysis now has the potential to analyse complex manufacturing processes such as welding. For example, it is possible to compute the transient temperature, displacement, strain, stress and microstructural changes during welding.^{[6]-[13]} However the capability is limited to the analysis of short, single pass welds. Improved algorithms are needed before automatic analysis of long or multipass welds becomes affordable for routine engineering.

The research needed to develop these algorithms covers a number of subjects. Three of the more important are: continuum mechanics for large deformation, large strain problems; element level error measures; and adaptive mesh management based on the previous two. Obviously several researchers are involved, all working toward the development of techniques which will lead to automatic and adaptive finite element procedures for the analysis of welding. The research also has wide application to other manufacturing processes.

Many manufacturing processes, including welding, have a nonlinear and time-dependent nature in which the areas of computational interest change and undergo large displacements. It is frequently impossible to use a uniform mesh which is fine enough to resolve the local phenomena, without yielding a number of unknowns which will overburden even the largest of today's supercomputers.^[14] There is a pressing need for efficient numerical modelling with the ability to perform dynamic, adaptive mesh modifications. The adaptive mesh-management algorithm will remove many of the time-consuming engineering operations related to mesh management. It will be particularly useful for time-dependent and nonlinear

problems in which the optimal mesh may change in every time step.

The research presented here addresses several concerns. First, recognizing the need to minimize the number of degrees of freedom (DOF) led to the development of elements that grade efficiently and without distortion. Second, to accommodate time-dependence and nonlinearities, an adaptive mesh-management algorithm was developed. The algorithm adaptively alters the spatial mesh in response to temporal changes providing a capability for automatic FEA. Finally, local error indicators are used to demonstrate that the algorithm responds to changing conditions. The response shows that the adaptive mesh-management algorithm will form an integral part of automatic FEA.

2. Isoparametric Elements for Mesh-Management Algorithms

Many finite element analyses are complicated by the presence of localized phenomena. For example, Figure 2.1a shows a uniform mesh surrounding a point load, Q . Figure 2.1b shows the most common form of graded mesh. However, this form of grading is inconvenient and results in element distortion which may reduce the accuracy of the solution.^{[15][16][17]} The type of grading desired for both 2-D and 3-D analysis is shown in Figure 2.1c. Meshes graded in this manner in the neighbourhood of localized phenomena such as point loads, point supports, re-entrant corners, crack tips and heat sources often provide dramatic computational savings.^[18] Although triangles and tetrahedrons can be used in mesh grading, many analysts prefer to use quadrilaterals and hexahedrons. The various approaches to this form of mesh grading give different names to the elements. They have been described as special rectangular elements, transition elements and contiguous elements.

The development of adaptive FEA also suggests that new mesh-grading elements are needed. Adaptive analysis using h-refinement; i.e., increasing the number of elements, requires effective mesh grading. In addition, the most serious attempts to develop fully automatic FEA have utilized graded meshes with spatial addressability.^{[2][5][19]} With the exception of the work by McDill^[20] and Cavendish,^{[21][22]} useful approaches to mesh grading with hexahedrons do not appear to have been published for 3-D FEA.

Mesh grading can be very effective providing that the cost of the finite element analysis is minimized and the accuracy of the solution is maximized. In short, for reasons of efficiency and computational economy, mesh-grading elements should:

- 1) minimize the number of degrees of freedom;
- 2) grade in a manner which minimizes the frontwidth or bandwidth of the problem;
- 3) have minimal element distortion;
- 4) maximize the subspace richness by using all available nodes;
- 5) permit spatial addressability.

Fig. 2.1 Meshes for 2-D analysis

a. Uniform mesh; b. Traditional grading; c. Preferred grading

It will be shown that the elements published by McDill^[20] and presented here, satisfy the criteria listed above. In addition, they are naturally suited to spatially addressable meshes and adaptive FEA.

2.1 Mesh-Grading Elements

Over the last two decades there has been considerable interest in mesh grading for 2-D analysis.^{[2][5][15][19]-[31]} However, a brief review of this literature will show that many of these approaches have limitations which reduce their effectiveness in 3-D mesh grading. The difficulties related to solving large matrices become critical in 3-D analysis, and it is for these analyses that the mesh grading is most important. Ideally, the approach selected for 3-D mesh grading should also be applicable to 2-D mesh-grading elements, which are still widely used. Even though the development of a new 2-D grading element offers little to the finite element community, it is a natural and concise means of illustrating the concepts required for 3-D elements. Both the 2-D and 3-D elements are computationally efficient, easily implemented in existing finite element programs and need not be distorted in the grading process. In addition, spatial addressability is preserved. Grading is accomplished by constraints which are embedded in the basis functions. The subspace richness is maximized by using all available nodes. To the author's knowledge, the new isoparametric hexahedrons are the first 3-D elements which can be graded efficiently and without distortion.

There are several distinct approaches to mesh grading which apply to meshes such as that shown in Figure 2.2. The nodes marked by open circle are obviously special. They are often referred to as irregular nodes. To maintain compatibility, some form of constraint^[30] is applied to the irregular node. In the most common approach, the irregular node is constrained so its behaviour is a linear function of the behaviour of the two adjacent vertex nodes.^{[2][19][24][26][29]} In effect, the constraint is applied to the two small elements rather than the one large element. The second constraint approach permits the irregular node to move. Thus, there are two linear fields on the edges of the small elements and the large element has a bilinear field on the adjoining edge.^{[20][25]} In other words, the larger element has the applied constraint. In either case, the constraint can be embedded in the element basis functions or applied during the factoring of the stiffness matrix.

If the mesh is composed of quadratic elements an alternative constraint approach can be used. A node outside the physical space of the element is allowed to replace the irregular

node.^{[23][28]} The coefficients of the basis functions for the element are then adjusted to reflect the out-of-position node. Finally, blending functions can be used.^{[21][22]} The unknowns for the irregular node are defined by a blending of the element edge functions for the nodes on the edges.

The selection of an appropriate approach to 3-D mesh grading was guided by the criteria listed above and by the desire to implement the approach in an existing finite element code, ASGAR¹. Since the essence of a finite element mesh is its set of global basis functions, all meshes which produce the same function space are, in this sense, identical. The selection of the approach to mesh grading was based on limiting the difficulties experienced by other researchers. For example, it was hoped that no special matrix operations would be needed for the global or element level stiffness matrices. It was decided that embedding the constraints in the basis functions is a superior approach. The reasons for selecting this approach to mesh grading will be examined more closely in the following material.

Kela^{[2][3][4]} and Rheinboldt^[19] use spatially addressable data structures for adaptive 2-D FEA using linear elements. These data structures accommodate the division of elements during refinement and the creation of new nodes. Irregular midedge nodes, such as those shown by open circles in Figure 2.2, and their associated corner nodes, are tagged. Constraint equations are applied to the irregular node so each of its variables is a linear function of the variables of its adjoining regular nodes. The stiffness matrix is then altered to reflect the constraints imposed by the irregular nodes. Somerville^[26] also uses constraint equations for the midedge nodes and applies additional special operations to the global stiffness matrix. The extra work required to add a degree of freedom and its constraint equation, and then manipulate the stiffness matrix to remove the inactive degree of freedom, is computationally inefficient. Accordingly, this approach was judged to be unsuitable. However, the spatially addressable data structures of Kela^{[2][3][4]} and Rheinboldt^[19] were of interest and will be dealt with later.

Walsh^[23] and Fortin^[28] use a special node labelling approach for grading 2-D quadratic elements. Walsh^[23] uses 8-node Serendipity elements whereas Fortin^[28] uses 9-node Lagrangian elements. In both cases the nodal list of a regular element is modified. Any of its

¹ ASGAR is a general purpose, nonlinear, transient finite element package developed at Carleton University.

Fig. 2.2 A graded mesh showing irregular nodes as open circles

midedge nodes which are nonconforming nodes in a neighbouring element, do not appear in the nodal list of the regular element. Grading is accomplished through a midedge node shift. Quite simply, a corner node from an adjacent element is used as the midedge node. This node is not in the physical space of the regular element. Special tagging procedures are required to indicate the replacement node and the regular edge nodes. The coefficients of the constraint equations for the nonconforming nodes are then used to modify the basis functions of the nodes on the affected edge.

This appeared to be very promising. However it was hoped that the special tagging procedures could be avoided. For this reason the basis functions for elements with shifted nodes were independently developed for both 2-D Lagrangian and Serendipity formulations. It was found that three distinct element types were required for effective grading. The first and second were needed for righthand and lefthand midedge node shifts. The third was needed when two midedge nodes were shifted on perpendicular, joining edges. The three element types can be thought of as having the shape of the letters p, q and w. Consequently, three new element types would need to be implemented and some way of recognizing the type of element would be required at the mesh generation stage.

The extension of the node-shifting approach to a 20-node brick and 3-D grading was found to be possible. However it involved corner node shifts, or alternatively the introduction of midface nodes, as well as midside node shifts. This difficulty and the multiple element types made this approach very cumbersome and less amenable to adaptive 3-D FEA. At that point, it was dismissed from further consideration.

The work of Cavendish^{[21][22]} is especially noteworthy because it clearly shows the possibility of extension to 3-D analysis. He presents piecewise linear polynomial finite elements with which local refinement or grading is readily handled for both 2-D and 3-D analysis. His nonstandard elements have linearly blended interpolation. The elements can be compatibly joined to any standard finite element. The unknowns; e.g., nodal displacements, are defined by a linear blending of the element edge or element boundary functions for the nodes on the edges. A set of interpolation functions can be created which are, in effect, a set of basis functions. When the standard elements are linear elements, the linear blending of the interpolation ensures compatibility at the element boundaries. Quadratic elements can also be used because the linear blending occurs with piecewise quadratic functions. Higher order elements which use cubically blended interpolation are also discussed for cases requiring

C^1 -compatibility.

It is unclear how Cavendish^{[21][22]} carries out the numerical integration. For example, he presents a linear element which has 4 nodes on one edge and 3 nodes on the other edge. The function is continuous, but irregular, and might require many Gauss points. Integration could be carried out explicitly over the physical space or by defining each linear segment to be a subregion for integration with modified Gaussian weights and sampling points. The implementation was ambiguous, although it appears to reduce to one very similar to that presented by Gupta^[25] and may be useful in developing a 3-D hierarchical element.

A recent contribution by Ang^[24] uses modified basis functions to constrain nodes and facilitate mesh grading. The constraints, whether for linear or quadratic elements, are applied to the regular elements. For example, two 4-node quadrilaterals can be joined to one larger 4-node quadrilateral. (In fact, three 4-node quadrilaterals can be joined to one, larger, 4-node quadrilateral.) The extra node does not appear in the larger element. Instead, the basis functions for the corner nodes, which lie on the joining line, are modified to reflect the constraint. Some form of tagging is required to distinguish an ordinary 4-node quadrilateral from one which has the modified basis functions. In addition, some sort of data structure is likely required to indicate the relationship between the constrained node and the other nodes on the joining line. The approach is suitable for isoparametric elements and permits very rapid grading. Although, Ang^[24] indicates that this approach can be extended to 3-D elements, no examples are given.

The technique for embedding the constraints in the basis functions is not a recent development. For example, in 1976, Carey^[31] demonstrated a similar technique for triangles using embedded constraints. An interesting use of embedded constraints was presented by Gupta in 1978.^[25] The constraints are embedded directly in the basis functions, no tags are required for irregular nodes and no special matrix operations are required. Rather than constraining a midedge node to the linear behaviour of the adjoining corner nodes, Gupta^[25] lets the midedge node be a regular degree of freedom. Thus the constraint is applied to the large element in which two linear integration fields are created instead of applying the constraints to the two smaller elements. Gupta^[25] also shows how Gaussian integration can be applied to the mesh-grading quadrilateral. Furthermore, this mesh-grading quadrilateral also satisfies the criteria listed above. Clearly, this is a superior approach and initial research quickly showed that it could be implemented in the existing

code and would be amendable to adaptive modelling.

Finally, a formulation such as Gupta's^[25] which showed positive results in the preliminary research, might also be used with Szabo's^[32] combined h- and p-refinement scheme. In the combined scheme the number of DOF is increased by both refinement and an increase in polynomial degree. It appears to be more difficult to do this with the quadrilaterals suggested by other researchers such as Walsh,^[23] Ang,^[24] Fortin^[30] and Somervaille^[26].

2.2 Mesh Grading in Two Dimensions

2.2.1 Formulation of the 4- to 8-Node Linear Quadrilateral

A new approach to the mesh-grading quadrilateral developed by Gupta^[25] is presented here. The intent is to introduce and clarify the concepts needed for mesh grading in three dimensions.

The 4- to 8-node grading quadrilateral is shown in Figure 2.3. Nodes 1 to 4 are mandatory. Nodes 5 to 8 are optional midedge nodes which may be added in any required combination. Midedge nodes may be treated as points of discontinuity in the gradient of a temperature or displacement field. No irregular nodes,^[19] constraint equations, or associated matrix operations such as those used by Somervaille^[26] are needed for the midedge nodes, since compatibility is inherent in the basis functions.

The basis functions for the 4- to 8-node quadrilateral are formulated in a straightforward manner using the standard rules for isoparametric elements:

$$\forall \quad P_i(\xi_j, \eta_j) = \delta_{ij} \quad (2.1)$$

Consider the application of a point load as was shown in Figure 2.2, with the mesh graded towards the point load. The nodes marked with open circles are common to three elements. They are corner nodes for the smaller elements (4,5 and 5,6) and midedge nodes for the larger elements (1,3). Element 1 is isolated in Figure 2.4a. The midedge node creates two smooth (C^∞ continuous) subelements within the element. Within each subelement the basis functions are linear. However, the basis functions for the midedge node and its two adjacent corner nodes are creased at the junction of the subelements and are, therefore, C^0 continuous. If node 5 were not present, element 1 would be a regular 4-node linear quadrilateral with linear basis functions; e.g., P_1^L , P_2^L . However, P_5 exists (Figure 2.4b),

Fig. 2.3 Numbering for the 4- to 8-node linear, isoparametric quadrilateral

therefore P_1 and P_2 must be 0 at node 5. The value of the linear basis function P_1^L is $1/2$ at node 5; the value of P_5 at node 5 is 1. Therefore P_1 (Figure 2.4c) and P_2 become:

$$P_1(\xi, \eta) = P_1^L(\xi, \eta) - 1/2P_5(\xi, \eta)$$

$$P_2(\xi, \eta) = P_2^L(\xi, \eta) - 1/2P_5(\xi, \eta)$$

where:

$$P_5(\xi, \eta) = 1/2(1 - |\xi|)(1 + \eta) \quad (2.2)$$

It is apparent that the presence of midedge nodes affects the basis corner nodes on that edge. For this reason, the basis functions (Table 2.1) should be evaluated, in order, from P_8 to P_1 . If a node is not present, it has no basis function. In effect, the value of the basis function and its derivatives are 0 everywhere. If a node is not present, its basis function and all subsequent references to the basis function should be removed from Table 2.1. Derivatives of the basis functions are presented in Appendix I.

Table 2.1 Basis Functions for the 2-D Grading Element

Node	$P(\xi, \eta)$
8	$1/2(1 + \xi)(1 - \eta)$
7	$1/2(1 - \xi)(1 - \eta)$
6	$1/2(1 - \xi)(1 - \eta)$
5	$1/2(1 - \xi)(1 + \eta)$
4	$1/4(1 + \xi)(1 - \eta) - 1/2(P_7 + P_8)$
3	$1/4(1 - \xi)(1 - \eta) - 1/2(P_6 + P_7)$
2	$1/4(1 - \xi)(1 + \eta) - 1/2(P_5 + P_6)$
1	$1/4(1 + \xi)(1 + \eta) - 1/2(P_5 + P_8)$

2.2.2 Numerical Integration in Two Dimensions

Normally Gaussian integration is carried out on C^∞ continuous functions, between the limits of -1 and +1. In the grading quadrilateral, the presence of midedge nodes with creased basis functions (C^0 continuous) makes the use of normal Gaussian integration inaccurate.

Fig. 2.4 4- to 8-node linear quadrilateral with one optional node
 a. Subelements created by optional node 5; b. Basis function for node 5;
 c. Basis function for node 1, when node 5 is present

A straightforward alternative is available. Integration may be carried out over the C^∞ continuous subelements within the element. The new regions for Gaussian integration in Figure 2.4a are: $-1 \leq \xi \leq 0$ and $-1 \leq \eta \leq 1$ for subelement 1, and $0 \leq \xi \leq 1$ and $-1 \leq \eta \leq 1$ for subelement 2. New sampling points and integration weights may be easily found as shown in Bathe.^[33] In effect, the integration can be performed by associating a separate transformation with each subelement. For integration from $-1 \leq \xi \leq 1$ and $-1 \leq \eta \leq 1$, a sampling point has co-ordinates (ξ, η) and weight W . For a rectangular subelement with limits $a \leq \xi \leq b$ and $c \leq \eta \leq d$, the co-ordinates of the new sampling point (ξ', η') and its weight W' are:

$$\begin{aligned}\xi' &= \left(\frac{a+b}{2}\right) + \left(\frac{b-a}{2}\right)\xi \\ \eta' &= \left(\frac{c+d}{2}\right) + \left(\frac{d-c}{2}\right)\eta \\ W' &= \frac{(b-a)(d-c)W}{4}\end{aligned}\tag{2.3}$$

For example, 2×2 integration in a 4-node linear quadrilateral becomes 2×2 integration in each of the 4 subelements in an 8-node linear grading quadrilateral.

2.2.3 Related Formulation

After completing the development of the basis functions and derivatives for both the 4- to 8-node quadrilateral and the corresponding 8- to 26-node hexahedron, a related approach by Guildenhuy^[34] was considered. He proposed that an additional node was required in the centre of the element; in other words, a 4- to 9-node quadrilateral. It quickly became apparent that the 4- to 8-node quadrilateral was a linear analogue of the 8-node quadratic Serendipity element. The 4- to 9-node quadrilateral was a linear analogue of the 9-node quadratic Lagrangian element. Before presenting the more complex 3-D mesh grading elements, it is important to examine the two formulations.

The condition numbers^[35] for the two element types were evaluated. Guildenhuy^[34] had not developed the basis functions or their derivatives for his 2-D formulation. However, the stiffness matrix for his element, with 9 nodes, can be assembled from four, 4-node linear quadrilaterals. The condition number is defined to be the ratio of the maximum and minimum eigenvalues of the stiffness matrix; i.e., $CN = \lambda_{max}/\lambda_{min}$, where $\lambda_{min} \neq 0$.

A bound on the number of significant figures that can be expected to be lost is, $SF \propto \log(CN)$.^[35] A high condition number, CN , indicates that larger errors are possible.

The condition numbers were calculated from the element(s) stiffness matrix. The nodes had 1 DOF; i.e., a thermal analysis was considered. The condition numbers are presented in Table 2.2. For convenience the condition numbers of quadratic Serendipity and Lagrangian elements are included. There is no significant difference in the condition numbers of the 4- to 8-node quadrilateral and the 4- to 9-node quadrilateral. A comparison of the eigenvectors gave no indication that either element was preferred, although the ninth node in the 4- to 9-node element does allow a tent-shaped eigenvector.

Table 2.2 Condition Numbers for Quadrilaterals

Element	λ_{max}	λ_{min}	CN
4 - to 8 - node linear, 8 nodes	4.000000	.5917518	6.76
4 - to 9 - node linear, 9 nodes	3.000000	.5917514	5.07
8 - node quadratic Serendipity	5.333333	.5104854	10.4
9 - node quadratic Lagrangian	6.605421	.5104854	12.9

When a 4- to 8-node quadrilateral (Serendipity analogue) has 4 subelements, the corner node basis functions become negative at the centre of the element as do those of a quadratic Serendipity element. It is well known that the finite element community prefers basis functions which are always positive. This is the major concern relating to the formulation of the 2-D and 3-D elements. The negative behaviour does not occur in the corresponding Lagrangian analogues.

There are also limitations in Lagrangian analogue. For example, the centre node is only required when 4 subelements are present. This condition can be generated by 4 combinations of 2 midedge nodes, or when there are 4 midedge nodes; i.e., once there are any 2 midedge nodes on nonparallel edges. Thus, it is not truly a 4- to 9-node quadrilateral because the nodes cannot be added in any desired combination. For example, there is no 5-node element with the fifth node at the centre, nor is there a 6-node element when the fifth and sixth nodes occur on nonparallel edges. This raises several concerns for the corresponding 3-D element. Once any two nonparallel midedge nodes are present on any surface, the midsurface node must be added. Further, once 8 subelements have been created,

a body-centred node is also required. In some cases, this will give rise to the requirement for more nodes to be present in the Lagrangian analogue than in the Serendipity analogue. For example, if midedge nodes are needed on three edges joining the same corner, the Lagrangian analogue would need 3 midsurface nodes and 1 body-centred node. These 4 nodes would not be needed in the Serendipity analogue. Furthermore, special techniques or data structures will be needed to implement the insertion of additional mandatory nodes. Certainly, the user cannot be expected to include all the mandatory nodes or to know when they are needed. Nevertheless, these are not insurmountable limitations, although they appear to restrict the flexibility of the element.

The decision to continue with the 4- to 8-node quadrilateral and its corresponding 8- to 26-node hexahedron rather than also developing the 4- to 9-node quadrilateral and its corresponding 9- to 27-node hexahedron was based on several factors. First, the condition numbers of the 2-D elements were similar. It is reasonable to assume the same would be true in the 3-D case. Second, a 9-node linear quadrilateral can be duplicated by using four, 4-node linear quadrilaterals. Similarly, eight, 8-node linear bricks can duplicate a 27-node linear brick. The additional complexity of implementing a 3-D adaptive algorithm for an 8- to 27-node brick, with midsurface and body-centred nodes becoming mandatory under certain situations was unattractive. Finally, Serendipity elements are more popular than Lagrangian elements,^[36] so it was felt that a Serendipity analogue might be easier for other researchers to implement.

Error estimates based on upper and lower bounds of the solution became available for 2-D elements in the late stages of the research for the thesis. It is appropriate then, to review the decision to develop only the Serendipity analogue. A fairly rigorous test of the Serendipity analogue will be useful in determining its behaviour relative to the Lagrangian analogue. Mashaie^[37] reports a *posteriori* element level error measures based on error bounds in the strain energy determined from the difference in the statically admissible and kinematically admissible stress fields. The elemental contribution to the relative error, ϵ_e , can be found from the sum of the subelement contributions, and is defined to be the elemental contribution to the error in the strain energy, Θ_e , divided by the total strain energy in the domain, E_0 .

$$\epsilon_e = \frac{\theta_e}{E_0} \quad (2.4)$$

The comparison of the two element types can again be based on four, 4-node elements, representing the subelements of the Lagrangian analogue, and one, 8-node Serendipity analogue element. The test was designed so the extra degree of freedom in the Lagrangian analogue was an active node and the advantage, if any, would be with the Lagrangian analogue. To ensure that there are no other negative basis functions in the problem, all other elements have only 1 or 2 subelements. The problem was 1 m $\times$ 1 m (with a thickness of 1 m). A compressive traction varying from 1×10^6 N/m² to 0 N/m² was applied at the top lefthand edge of the first subelement. A plane stress analysis was performed.

Each element's contribution to the relative error and the total relative error for each mesh are tabulated (Table 2.3) for the mesh shown in Figure 2.5. Examples of the projected stresses are shown in Figures 2.6.

Although the total relative error is lower in the mesh with the Serendipity analogue, it also has more strain energy. In both cases, as expected, the error and the strain energy are concentrated in the element under the traction. For element 1, the contribution to the error is lower with the Lagrangian analogue. Thus, it is clear that there is some advantage to using the Lagrangian analogue in the region of the applied load. However, it also seems as if the Lagrangian analogue has polluted the solution slightly for surrounding elements. The stress projections are very similar in the vicinity of the load, and are only slightly different elsewhere. Although it appears as if the Lagrangian analogue offers better solutions in the vicinity of a singularity, it is not a tremendous improvement.

In future research, it might be useful to formulate the basis functions for the 4- to 9-node Lagrangian analogue element to examine its behaviour when it is partially filled. In the 3-D case the comparison will be more difficult to make, due to the number of possible combinations of mandatory nodes (midsurface nodes and body-centred nodes). Any difficulties in formulating the 3-D element will extend into the adaptive algorithm which is presented later. One particular concern is how the mandatory nodes will come into play in a propagation problem, when refining one element means that its neighbours also have to be refined. Nevertheless, developing the special data structures needed to support such propagation and check for the addition of mandatory nodes does not appear

Fig. 2.5 Mesh for comparison of element types

- - - shows subelement division

a

b

Fig. 2.6 Projected Stresses σ_{yy} , MPa

a. Lagrangian Analogue; b. Serendipity Analogue

Contours: A = -0.5; B = -0.4; C = -0.3; D = -0.2; E = -0.1; F = 0

to be an overly difficult problem. In the meantime, if there is concern about an area near a singularity, simply using eight, 8-node bricks, in place of a 26-node brick, will produce the additional degrees of freedom and prevent basis functions from having negative values.

Table 2.3 Elemental Contribution to Relative Errors, ϵ .

Element	Serendipity Analogue	Lagrangian Analogue
1	.0371	.0165
2	.0050	.0084
3	.0116	.0202
4	.0012	.0022
5	.0248	.0444
6	.0145	.0260
7	.0027	.0048
<i>total relative error</i>	.0969	.1226
<i>strain energy</i>	.0189	.0105

2.2.4 Implementation

The emphasis of this thesis is in 3-D finite element analysis. However, the 4- to 8-node quadrilateral has been implemented in ASGAR for 2-D heat transfer, 2-D stress analysis and 2-D thermal-elasto-plasticity. Once the elements were implemented for these analyses, Meekisho^[38] implemented the quadrilaterals for 2-D contact mechanics for thermal analysis and stress analysis.

The implementation was quite general and straight-forward. With a few exceptions the implementation follows the procedures common to all other element types in ASGAR. Obviously, the most significant changes required for the 4- to 8-node quadrilateral are those associated with calculating the element level stiffness matrix. Accordingly, the changes to ASGAR were essentially limited to the routines which calculate the element level stiffness matrix. Since a frontal solver is used, there is no need to assemble a global stiffness matrix. Four new subroutines were required to accommodate the following:

1. compaction of the topological nodal lists (COMPACT);
2. the new basis functions and their derivatives (SFHQD);
3. calculation of the number of subelements and their integration limits (SUBDOM2D);

4. calculation of Gaussian integration points and weights (GAUSH2D);

It is expected that the mesh-grading elements could be implemented as easily in many finite elements codes.

COMPACT is needed because ASGARD was originally written for elements with a fixed number of nodes. A compaction routine was the most effective way of achieving compatibility with the existing Do-Loop limits. For example, in ASGARD the element level load vectors are assigned in a Do-Loop over the number of nodes in the element (LNODZ). In a graded element, there might be 5 nodes but these might be in positions 1 to 4 and 8 in the element topology (LNODS). A loop over the number of nodes would stop at the fifth position in the topology list. It was a simple matter to use a compacted list of nodes. With compaction, the value of the basis function for any fifth node can be found in the fifth position in a compacted list. This was also very important in any part of the code which used the basis functions or their derivatives in loops over the number of nodes in the element. The element topology, in terms of local node numbering, is also determined in COMPACT. As in the full topology, non-existent nodes are indicated by a 0. This vector is used to indicate which basis functions are active in the element.

In SUBDOM2D, the local topology of each element determines the number of subelements and their respective integration limits. These limits are used in GAUSH2D to determine the total number of Gauss points and their weights. The routines which calculate the stiffness matrices are FT2D for heat transfer and PLAS2D for stress analysis. These routines loop over the number of Gauss points calling SFHQD to evaluate the basis functions and their derivatives for each Gauss point. In SFHQD the local topology is used to determine which nodes are active in the element being considered.

A number of other changes were made to the code. Many of these can be classified as global improvements to ASGARD. For example, to accommodate the variable nature of the number of nodes, some arrays were rearranged so the fixed dimension appeared first. Typically this involved placing the dimension of the domain first; e.g. planar problems are two dimensional. The second dimension is the variable number, such as number of nodes. Point, edge and surface (body) loads can be used with the mesh-grading elements. In edge loading two additional subroutines were developed. The first deals with the one-dimensional (1-D) basis functions and their derivatives. The second contains the Gauss

points and weights for 1-D integration.

Variable node elements such as the 4-to 8-node quadrilateral are not standard elements for postprocessors. TAURUS,^[39] the postprocessor used at Carleton is able to plot 4-node linear elements and their associated fields. Therefore special routines were developed to implement the graphics the mesh-grading elements. To utilize TAURUS,^[39] it was necessary to resolve the 4- to 8-node quadrilaterals in a graded mesh into their 4-node subelement components. Obviously, in some cases this introduces new nodes. The co-ordinates of these nodes can be interpolated by summing, over the nodes, the product of the basis function, evaluated at the point, and the value of the function at the node. Temperatures and displacements can be found the same way. This is usually expressed as:

$$T = \sum P_i T_i \quad (2.5)$$

At this point the advantages of spatial addressability became evident. Preserving spatially addressability in the mesh allowed a spatially addressable data structure to be used to keep track of the identities of the new nodes as well as their temperatures and displacements. The k-D tree implemented as an abstract data type, was used. It will be discussed in more detail later. A more comprehensive discussion of associated developments for graphics is presented in Appendix II.

The elements have been used most frequently in computational weld mechanics. An example of this is given by Goldak.^[8]

2.3 Mesh Grading in Three Dimensions

2.3.1 Formulation of the 8- to 26-Node Linear Hexahedron

The 8- to 26-node linear hexahedron (Figures 2.7a and 2.7b) is formulated much like the grading quadrilateral and also satisfies the criteria outlined earlier. Nodes 1 to 8 are mandatory. Nodes 9 to 20 are optional midedge nodes, nodes 21 to 26 are optional midface nodes. The optional nodes may be used in any desired combination. There is no need for element distortion when the hexahedrons are used in graded meshes and no constraint equations are required in the global equations.

The midface and midedge nodes create smooth (C^∞ continuous) subelements within the element. Within each subelement the basis functions are linear. However, the basis functions for the midface nodes and midedge nodes as well as their adjacent corner nodes, are creased at the junction of the subelements and are, therefore, C^0 continuous. Consider, for example, the element in Figure 2.8 with 2 optional nodes in the positions of nodes 10 and 25.

$$P_{25} = 1/2(1 - |\xi|)(1 - |\eta|)(1 + \zeta) \quad (2.6)$$

P_{10} must be 0 at node 25, therefore:

$$P_{10} = 1/4(1 - \xi)(1 - |\eta|)(1 + \zeta) - 1/2P_{25} \quad (2.7)$$

P_3 must be 0 at both node 10 and node 25, therefore:

$$P_3 = 1/8(1 - \xi)(1 - \eta)(1 + \zeta) - 1/2P_{10} - 1/4P_{25} \quad (2.8)$$

The presence of a midface node alters the basis functions of those midedge nodes present on that face as well as the corner nodes on that face. The presence of a midedge node, in turn, alters the basis functions of the corner nodes on that edge. For this reason, the basis functions (Table 2.4) should be evaluated, in decreasing order, from P_{26} to P_1 . If a node is not present, its basis function and all subsequent references to it should be removed from Table 2.4. Derivatives of the basis functions are presented in Appendix II.

a

b

Fig. 2.7 Node numbering for the 8- to 26-node linear, isoparametric hexahedron
 a. Vertex nodes and optional midedge nodes; b. Optional midface nodes

Fig. 2.8 8- to 26-node hexahedron with 2 optional nodes

Table 2.4 Basis Functions for the 3-D Grading Element

Node	$P(\xi, \eta, \zeta)$
26	$1/2(1 - \xi)(1 - \eta)(1 - \zeta)$
25	$1/2(1 - \xi)(1 - \eta)(1 + \zeta)$
24	$1/2(1 - \xi)(1 - \eta)(1 - \zeta)$
23	$1/2(1 - \xi)(1 + \eta)(1 - \zeta)$
22	$1/2(1 - \xi)(1 - \eta)(1 - \zeta)$
21	$1/2(1 + \xi)(1 - \eta)(1 - \zeta)$
20	$1/4(1 + \xi)(1 - \eta)(1 - \zeta) - 1/2(P_{21} + P_{24})$
19	$1/4(1 - \xi)(1 - \eta)(1 - \zeta) - 1/2(P_{22} + P_{24})$
18	$1/4(1 - \xi)(1 + \eta)(1 - \zeta) - 1/2(P_{22} + P_{23})$
17	$1/4(1 + \xi)(1 + \eta)(1 - \zeta) - 1/2(P_{21} + P_{23})$
16	$1/4(1 + \xi)(1 - \eta)(1 - \zeta) - 1/2(P_{21} + P_{26})$
15	$1/4(1 - \xi)(1 - \eta)(1 - \zeta) - 1/2(P_{24} + P_{26})$
14	$1/4(1 - \xi)(1 - \eta)(1 - \zeta) - 1/2(P_{22} + P_{26})$
13	$1/4(1 - \xi)(1 + \eta)(1 - \zeta) - 1/2(P_{23} + P_{26})$
12	$1/4(1 + \xi)(1 - \eta)(1 + \zeta) - 1/2(P_{21} + P_{26})$
11	$1/4(1 - \xi)(1 - \eta)(1 + \zeta) - 1/2(P_{24} + P_{26})$
10	$1/4(1 - \xi)(1 - \eta)(1 + \zeta) - 1/2(P_{22} + P_{26})$
9	$1/4(1 - \xi)(1 + \eta)(1 + \zeta) - 1/2(P_{23} + P_{26})$
8	$1/8(1 + \xi)(1 - \eta)(1 - \zeta) - 1/2(P_{15} + P_{16} + P_{20}) - 1/4(P_{21} + P_{24} + P_{26})$
7	$1/8(1 - \xi)(1 - \eta)(1 - \zeta) - 1/2(P_{14} + P_{15} + P_{19}) - 1/4(P_{22} + P_{24} + P_{26})$
6	$1/8(1 - \xi)(1 + \eta)(1 - \zeta) - 1/2(P_{13} + P_{14} + P_{18}) - 1/4(P_{22} + P_{23} + P_{26})$
5	$1/8(1 + \xi)(1 + \eta)(1 - \zeta) - 1/2(P_{13} + P_{16} + P_{17}) - 1/4(P_{21} + P_{23} + P_{26})$
4	$1/8(1 + \xi)(1 - \eta)(1 + \zeta) - 1/2(P_{11} + P_{12} + P_{20}) - 1/4(P_{21} + P_{24} + P_{26})$
3	$1/8(1 - \xi)(1 - \eta)(1 + \zeta) - 1/2(P_{10} + P_{11} + P_{19}) - 1/4(P_{22} + P_{24} + P_{26})$
2	$1/8(1 - \xi)(1 + \eta)(1 + \zeta) - 1/2(P_9 + P_{10} + P_{18}) - 1/4(P_{22} + P_{23} + P_{26})$
1	$1/8(1 + \xi)(1 + \eta)(1 + \zeta) - 1/2(P_9 + P_{12} + P_{17}) - 1/4(P_{21} + P_{23} + P_{26})$

2.3.2 Numerical Integration in Three Dimensions

A maximum of 8 subelements can be created in this element, each with its own integration limits. For integration from $-1 \leq \xi \leq 1$, $-1 \leq \eta \leq 1$ and $-1 \leq \zeta \leq 1$, a sampling point has co-ordinates (ξ, η, ζ) and weight W . For a subelement (hexahedron) with limits $a \leq \xi \leq b$, $c \leq \eta \leq d$ and $e \leq \zeta \leq f$, the co-ordinates of the new sampling point (ξ', η', ζ') and its weight W' are:^[33]

$$\begin{aligned}\xi' &= \left(\frac{a+b}{2}\right) + \left(\frac{b-a}{2}\right)\xi \\ \eta' &= \left(\frac{c+d}{2}\right) + \left(\frac{d-c}{2}\right)\eta \\ \zeta' &= \left(\frac{e+f}{2}\right) + \left(\frac{f-e}{2}\right)\zeta \\ W' &= \frac{(b-a)(d-c)(f-e)W}{8}\end{aligned}\tag{2.9}$$

2.3.3 Implementation

The 8- to 26-node hexahedron has been implemented in ASGARD for 3-D heat transfer and 3-D stress analysis. Oddy^[40] implemented the elements for 3-D thermal-elasto-plasticity. In addition, Meekisho^[38] implemented the hexahedrons for 3-D contact mechanics in both thermal and stress analyses.

Again, the implementation was quite general and straight-forward. It followed the procedures common to all other 3-D element types in ASGARD. The most significant changes required for the 8- to 26-node hexahedron were those associated with calculating the element level stiffness matrix. Three new subroutines were required to accommodate the following:

1. the new basis functions and their derivatives (SFHBR);
2. calculation of the number of subelements and their integration limits (SUBDOM3D);
3. calculation of Gaussian integration points and weights (GAUSH3D).

The compaction of the nodal lists and generation of the local topology used the routine COMPACT as did the 2-D elements.

In SUBDOM3D, the topology of each element determines the number of subelements and their respective integration limits. These limits are used in GAUSH3D to determine the total number of Gauss points and their weights. SFHBR then evaluates the basis functions

and their derivatives for each Gauss point, in a loop over the number of Gauss points. The routines which calculate the stiffness matrices are FT3D for heat transfer and ETPL3D for stress analysis. These routines loop over the number of Gauss points calling SFHBR to evaluate the basis functions and their derivatives for each Gauss point. In SFHBR the topology is used to determine which nodes are active in the element being considered.

Point and volume loads can be used. As with the 2-D mesh-grading elements, the spatial addressability of the mesh was exploited during the development of special routines to implement the graphics. The advantages were even more evident in three dimensions in which the possible combinations of subelements can be quite complex. A brief discussion of associated developments for graphics is presented in Appendix III.

One limitation in the current implementation of the 3-D elements is that they cannot yet be used for surface loading such as convection. For other 3-D element types in ASGARD, the user must correctly enter the nodes over which the surface loading is desired. The surface is identified by its nodes and the loads are integrated over the surface using the basis functions of the 2-D element which matches the surface. For example, the 8-node quadratic quadrilateral is used with the 20-node quadratic brick. There are two possibilities for surface loading for the 3-D mesh-grading elements: locally defined surface loading such as that used for the other element types, and globally defined loading. Globally, surfaces which lie in a specific plane, or centroids of surfaces which lie on a selected curve could be tagged for identification. An automatic facility within the code would provide the proper loading at the local level. In either case, at the local level, the 4- to 8-node quadrilateral, can be used for surface loading of an 8- to 26-node hexahedron as long there are no midsurface nodes. If there were midsurface nodes, integration could occur over four, 4-node quadrilaterals. This requires some capability to identify the surface of the element in question. Surfaces can be numbered with respect to the midsurface nodal positions; e.g., the surface which would take node 21 (in the local space of the element) could be identified as the surface 1. Similarly, the surface which would take node 26 could be called surface 6. Code could be implemented so the user identified the surface identity explicitly, and the node numbers on the face were determined by the code. Alternatively, the user could input the corner nodes (vertices) of the face. Data blocks could then be used to identify all possible, legitimate, combinations of node numbering for the surface. These concepts were suggested to Mashaie,^[87] who has now implemented a limited version of surface loading in his code.

The 3-D mesh-grading elements have been used most frequently in computational weld mechanics. To the author's knowledge the first transient, 3-D thermal-elasto-plastic analysis of a weld was published by Goldak.^{[8][9][10]} The grading elements were instrumental in the analysis.^[13] It seems that analyses of this type are not routine for other researchers because adequate models have not been available.

2.3.4 Validation of the 8- to 26-Node Hexahedron

The benefits of using the 8- to 26-node hexahedron for modelling problems with localized phenomena, can be demonstrated by comparing finite element analyses performed using uniform and recursively graded meshes. The problem considered here is that of a point thermal load in an infinite, isotropic body for which the exact solution for the steady state temperature T is:^[41]

$$T = \frac{Q}{4\pi k R} + T_{\infty} \quad (2.10)$$

The following parameters were used: $Q = 20,000$ W, $k = 1$ W/m°C, $T_{\infty} = 20^{\circ}\text{C}$.

The problem was solved using both uniform and recursively graded finite element meshes. The latter is shown in Figure 2.9. In all cases symmetry was used to reduce the problem to an octant. To establish the boundary conditions a $10\text{ m} \times 10\text{ m} \times 10\text{ m}$ octant was selected for analysis. Temperatures from the exact solution were applied as the boundary conditions for the external faces. The three faces on planes of symmetry were insulated. The point thermal load Q , applied to all finite element models was 2500 W. ASGARD, a frontal solution finite element code was used. All analyses were performed on a 0.2 MIPS, Apollo DN320 workstation.

Figure 2.10 shows a comparison between the exact solution and the solutions obtained using the two mesh types. With an $8 \times 8 \times 8$ uniform mesh the problem was solved in 1840 cpu seconds. With the equivalent recursively graded mesh the same problem was solved in 58 cpu seconds, with no perceptible loss in accuracy.

In this instance, the results were compared to the analytical solution. The adaptive algorithm mesh-management algorithm presented later will eventually be guided by appropriate and rigorously generated error measures.

The mesh-grading elements are especially useful for problems which have localized phenomena.^[42] The author has found that this approach to mesh grading is remarkably effective for computational weld mechanics in which the heat source is highly localized.^{[8][9][10]}

Fig. 2.9 Graded 3-D mesh with 3 levels of recursion

Fig. 2.10 Temperature vs radius for the $8 \times 8 \times 8$ uniform mesh and graded mesh with 3 levels of recursion

— analytical solution; ○ uniform mesh; ■ graded mesh.

Figure 2.11 shows an example of a displacement field obtained from a nonlinear, transient, thermo-elasto-plastic FEA of a tack weld in a butt joint. Without the grading elements, the cost of this analysis would have been prohibitive.^[10] Only 144 grading elements (304 nodes) were required to analyse a 2.3 cm weld in an AISI 1020 carbon steel. The arc travelled at 0.508 cm/s in 29 time steps with a power source of 1.7 kW ^[9].

2.4 Computational Effectiveness of Mesh-Grading Elements

The new grading elements offer several advantages over traditional constraint techniques. With the new elements, all degrees of freedom are active and no constraint equations or associated matrix operations are needed. The extra work required to add a degree of freedom and its constraint equation, and then manipulate the stiffness matrix to remove the inactive degree of freedom, is wasteful. These steps are not required with the new elements. Furthermore, since no constraint equations are needed, additional data structures to represent the constraint equations are avoided. For these reasons, it is felt that the new grading elements simplify procedures. The global matrix obtained with the grading elements will be the same as that obtained with constraint techniques such as that used by Somerville^[26] provided the appropriate nodes are constrained. For example, the 2-D meshes shown in Figure 2.12 will give the same global stiffness matrix if node X in 2.12b is constrained. However, in view of the advantages mentioned above, it is felt that the new grading elements are superior.

Commercial finite element codes typically use a direct solver. With such a system, factoring the stiffness matrix is the most costly portion of a finite element analysis. This cost, C_f , is proportional to the number of degrees of freedom in the problem, d_f , and to the square of the frontwidth, f_w (or bandwidth):

$$C_f \propto d_f \times (f_w)^2 \quad (2.11)$$

If r is the number of recursions in a graded mesh and n is the number of elements on the edge of a uniform mesh, then for a graded mesh in which the size of the smallest element is the same as those in the uniform mesh, the relationship between n and r is:

$$n = 2^r \quad (2.12)$$

Fig. 2.11 Displacement field for mesh-grading elements used in computational weld mechanics
Displacement scale factor = 50

a

b

Fig. 2.12 Graded meshes with the same global stiffness matrix.
a. Mesh using 4- to 8-node quadrilaterals
b. Mesh using 4-node quadrilaterals and a constraint for node X

Factoring the stiffness matrix dominates the cost of the analysis in problems with large n and r . Hence, for an implicit analysis of a point load on a 3-D uniform mesh, the computational complexity is $O(n^7)$. For the equivalent recursively graded mesh the computational complexity is $O(\log_2 n)$. The corresponding 2-D analyses have computational complexities of $O(n^4)$ and $O(\log_2 n)$. Table 2.5 shows C_f for the uniform and recursively graded meshes. Note, for heat transfer problems, $D = 1$; for problems in stress analysis $D = 3$.

Table 2.5 Cost of Factoring the Stiffness Matrix

Description	f_w	d_f	C_f
Uniform mesh : n elements, $n + 1$ nodes on an edge	$(D(n + 1))^2$	$D(n + 1)^3$	$O(n^7)$
Recursively graded mesh : $r = \log_2 n$, fixed front width	$\propto D$	$f_w \times r$	$O(\log_2 n)$

Grading in this fashion, using direct formulation of the basis functions, rather than using a uniform mesh, represents a considerable reduction in computational complexity, particularly for 3-D analysis. It is this advance in 3-D mesh grading that enabled a 1.5 MIPS, \$30,000 Apollo DN3000 workstation to perform 3-D transient thermal-elasto-plastic analysis of welds, faster than today's supercomputers using traditional analysis.^[44] However, the advantages gained by such grading are approaching a practical limit in current 3-D thermal-elasto-plastic analyses. Consider the temperature and displacement fields shown in Figures 2.13 and 2.14 for a 5 cm butt weld in a plate 0.635 cm thick. Three-dimensional thermal-elasto-plastic analysis for this problem takes nearly 60 cpu hours on an Apollo DN3000. There are 186 elements in the mesh, of these, 128 are contained in the highly refined region along the path of the source.

In Figure 2.14 it is obvious that the displacement field is changing very rapidly over very short distances perpendicular to the plane of symmetry. Lines which were originally perpendicular to the plane of symmetry may bend forward or backward but at the plane of symmetry, they must always be perpendicular to that plane. If they are not perpendicular it implies there are shear forces on the plane of symmetry. This is not possible. Clearly the mesh is too coarse. Furthermore, the stress field present indicates that the weld may be too short.^[40] Thermal-elasto-plastic analysis of a longer weld with a finer mesh would be prohibitively expensive. It is this limit which led to the development of an adaptive

mesh-management algorithm. The algorithm will allow intensive grading to surround the heat source, and, in effect, move with the source in a much longer weld.

Fig. 2.13 Temperature field for a butt weld
A = 200°C; B = 400°C; C = 600°C; D = 800°C;
E = 1000°C; F = 1200°C; G = 1400°C; H = 1600°C.

Fig. 2.14 Displacement field for a butt weld

Displacement scale factor = 50

3. An Adaptive Mesh-Management Algorithm

Today, although FEA programs are widely used by engineers, they are rarely used by designers. It is expected that automatic FEA will soon enable designers to analyse and test the viability of their designs. This has the potential to significantly shorten the product development cycle. However, many have taken the position that automatic FEA would amount to a black box which would be too dangerous for designers to use. The eventual goal is that automatic FEA should be as trustworthy and as safe as other mathematical functions used in everyday engineering. One such everyday function is the square root. Interest in automatic FEA has been developing over the last decade.^{[27][45]} During this time, two principal issues have been identified: adaptive mesh-management procedures and *a posteriori* estimates of the error. Error estimates measure the error in an analysis and provide the feedback needed to steer the analysis towards a more accurate solution. The adaptive procedures manage or alter the mesh in response to the error estimates.

In the analysis of manufacturing processes such as welding, many phenomena are localized and transient. Adaptive mesh management based on error bounds or estimates will be a powerful tool for dealing with moving singularities and other types of irregular behaviour. Consider the approach to grading a mesh as shown in Figures 2.13 and 2.14 for the analysis of a weld. As indicated previously, grading in this fashion, rather than using a uniform mesh, represents a considerable reduction in computational complexity. However, the advantages gained by such grading are approaching a practical limit. A 3-D thermal-elasto-plastic analysis of this 5 cm weld, takes nearly 60 cpu hours on a 1.5 MIPS, Apollo DN3000 workstation. A mesh, graded in all directions and changing with time; i.e., adaptive and dynamic mesh management, will reduce the direct involvement of the engineer and provide computational savings.^[18] To illustrate this, consider a simple cost comparison based on doubling the length of a bar in an analysis of a weld. In a static mesh, both the number of degrees of freedom and the number of time steps required, are proportional to the length. Doubling the length, L , doubles the number of degrees of freedom, which increases the cost of factoring the stiffness matrix by $O(L)$. Doubling the length also doubles the number of time steps at a cost of $O(L)$. Thus, for a static mesh, doubling the length increases the total cost by $O(L^2)$. In a dynamic mesh, the length of the bar can be changed substantially without significantly altering the number of degrees of freedom. This can be

assigned a cost approaching $O(1)$. The number of time steps is still proportional to the length L . Therefore, for a dynamic mesh, doubling the length L doubles the total cost. It is quite clear that as larger, more realistic problems are analysed, the use of a dynamic mesh will be critical. Ultimately, such adaptive procedures will enable the analysis of multipass welds and other similar manufacturing processes by automatically adapting the finite element mesh for the existing and expected conditions.

There are five basic requirements for a complete automatic FEA system:^{[1]-[4]}

1. The user should only specify the geometry of the object, not its finite element mesh.
2. The related finite element mesh should be generated automatically.
3. There should be an appropriate element library and equation solution procedure.
4. Mesh management should be controlled by some form of rigorously generated error estimates.
5. Schemes for adapting and improving the finite element model should be present.

Several of the above items are already in place or under development as separate research at Carleton. PADL-2^[46], a solids modeller, and INGRID^[47], which supports higher level operations on entities such as splines, are input data structures which automatically generate finite element meshes. ASGARD has an existing element library which now includes the mesh-grading elements presented earlier. The equation solution procedure is well established in ASGARD. A direct solver using a frontal solution technique is applied. ASGARD is based on sound physics for 3-D transient, nonlinear, large strain, large deformation thermal-elasto-plastic stress analysis.^[40] Error estimates which will eventually guide the adaptive algorithm are being developed by Mashaie.^[37] Of course, data must be properly transferred as the mesh changes. Goldak^[48] is developing techniques for data transfer. However, without the ability to adaptively alter the mesh, automatic FEA will not be possible. Thus, the remaining, and very crucial area, is that of adaptive procedures for mesh management.

Clearly, it is critical to develop a new adaptive algorithm which will alter a finite element mesh in response to rigorously generated *a posteriori* error estimates. The mesh-grading elements presented earlier will be the keystone of the new algorithm. The adaptive mesh-management algorithm which will be introduced is rule-based and makes use of array data structures and the spatially addressable k-D tree,^[48] implemented as an abstract data type.

These adaptive procedures are designed to produce dynamic and hierarchical finite element meshes. This new adaptive algorithm, when used in combination with other research at Carleton, will provide an essential step towards fully automatic FEA.

3.1 Dynamic Mesh Management

The adaptive mesh-management algorithm should be quite general and capable of functioning with many finite element packages. Nevertheless, the first productive, fully automatic and adaptive finite element analysis will be realized more quickly if existing codes and capabilities are incorporated where possible. For this reason, it was decided to utilize ASGARD, a general purpose, 3-D, nonlinear, transient, thermo-elasto-plastic finite element package available at Carleton. Although this places certain constraints on the algorithm, they are not overly restrictive and will be mentioned where appropriate. The first implementation of the adaptive algorithm will be limited to transient heat transfer and small deformation, small strain, stress analysis in which the nodal positions do not change appreciably. The extension to large deformation, large strain problems adds a certain degree of complexity to the algorithm and will be considered later, as a separate issue.

It is well known that the essence of FEA is to project the exact solution from an infinite dimensional function space (Hilbert space) onto a subspace of finite dimension. This reduces computing costs to affordable levels. To be effective, the subspace must be designed so the errors in the projection are acceptably small. Since creating the mesh specifies this subspace, a good mesh is critical to accurate and efficient FEA. If the errors in the projection are too large, it is necessary to modify the mesh to reduce the errors. The error measures specify the areas where modifying the mesh will reduce the errors most effectively.

There are three distinct and well-understood approaches to adaptively modifying a finite element mesh. The first is to enlarge the function space. This can be accomplished through either h-refinement; i.e., increasing the number of elements; or p-refinement; i.e., increasing the polynomial order of the elements. Combined h- and p-refinement can also be used. The second, involves deforming the function space by moving the nodes. This group is classified as moving finite element (MFE) methods. In static problems the nodes are moved to reduce the error. In transient problems the nodes are moved to reduce the error in tracking some physical phenomenon. Finally, an entirely new mesh can be created.

This is seen most often in cases dealing with large deformations and is sometimes called rezoning. It is anticipated that automatic FEA will eventually involve the sequential use of all three methods.

In p-refinement the degree, p , of the polynomial shape functions is increased for some elements in a mesh with a fixed number of elements.^{[49]-[52][56]} For example, Peano^{[49][50]}, Katz^[51] and Babuška^[52] use adaptive p-refinement for C^0 and C^1 triangular elements with area co-ordinates. In this family of elements, basis functions corresponding to an element of degree p are a subset of those corresponding to an element of degree $p + 1$. Thus, the stiffness matrix of the initial element (degree p) is a submatrix of the stiffness matrix of the enriched element (degree $p + 1$). This means that when the order of the element is increased from p to $p + 1$, only the new rows and columns of the stiffness matrix need to be determined. A determination of the convergence of the solution is based on an estimate of the error in the strain energy.

H-refinement refers to a reduction in the size, h , of some of the elements in the mesh. The number of elements increases and the accuracy of the solution improves. H-refinement is used extensively in many adaptive finite element algorithms.^{[18][53][54]-[60]} For example, Bank^{[57]-[60]} discusses adaptive grid refinement for elliptic partial differential equations for 2-D problems. The associated software is known as the PLTMG package.^{[58][60]} In this scheme, mesh refinement is guided by the error in the energy norm as developed by Babuška.^[60] For triangular elements localized h-refinement is performed by nested triangulation in which new elements are formed by dividing the linear parent element. In this technique, nonconforming or irregular nodes can be introduced.^{[62][63]} Such nonconforming nodes are accommodated by connecting them to the opposite vertex of the element they are in, and subsequently creating two more elements.

Moving mesh finite element methods are techniques in which the nodes are moved during an analysis. Its power lies in the fact that a near optimal grid, for the given mesh topology, can be obtained with the solution. Some elements are stretched or distorted, so other elements become fine elements and reduce local errors. As such, the nodes move away from areas in which there is little activity and towards areas where a greater nodal density is required.^{[70]-[73]} In automatic FEA it is expected that MFE methods could be used for short periods following h- and/or p-refinement. The MFE methods could be used to track internal boundaries such as those associated with phase changes or shock waves.

Unfortunately, some moving finite element methods cause nodes to pile up. This can lead to unduly deformed elements.^{[74][75]} Although it is possible to avoid this, the computational costs appear to be severe.^[76] In general, MFE methods did not appear to be an appropriate approach at this time.

It is expected that rezoning will ultimately follow MFE methods in adaptive FEA. Complete rezoning would occur once a large numbers of elements in the mesh exhibited some predetermined level of deformation. For example, Oddy's^[17] distortion metric could be used to evaluate element level distortion prior to rezoning. However, rezoning can also be used for large deformation FEA. In many problems, especially those relating to metal forming in which large deformations are observed, the need to rezone is unavoidable since Lagrangian formulations lose accuracy as the mesh distorts.^{[15][16][17][77]} A FEA mesh is said to be Lagrangian when the reference system coincides with the material co-ordinate system.^[78] Hence any node remains coincident with a given particle of material and each element will contain the same material throughout the problem. Obviously, correct transfer of data from the old mesh to the rezoned mesh is a critical part of rezoning. Data transfer and large deformation finite element formulations will be discussed later.

The first stage in the development of the new adaptive algorithm was to decide whether h-refinement, p-refinement or combined h- and p-refinement should be used. Several considerations led to the choice of h-refinement. First, many of the concepts for dividing the elements in a manner similar to h-refinement had been established during the development of the graphics routines for the 3-D elements. Second, it is well understood that if the solution is rough; e.g., the gradient is discontinuous, then h-refinement is more efficient.^{[32][79][80][81]} This often occurs at liquid-solid and elasto-plastic boundaries as well as in the vicinity of point loads. A welding heat source is considered to be a weak singularity, therefore h-refinement is preferred near the heat source. Other researchers^{[27][32][81][82]} have shown that combined h- and p-refinement provides the best approach to adaptive algorithms particularly if h-refinement is used in areas where the solution gradient changes rapidly. Although several researchers have compared different approaches to refinement, there appears to be very little discussion of coarsening. Certainly there are no comparisons of h-coarsening and p-coarsening in the available literature.

Refinement, of course, improves accuracy. However, maintaining a fully refined mesh can be expensive, thus coarsening, can be used in turn, to reduce the computational costs.

Since coarsening typically follows refinement, a clear way to amalgamate the refined elements is required; i.e., the routines must be able to identify the previously defined parent element or, alternatively, select a new parent. In p-coarsening this amounts to removing nodes from the element. Conceptually h-coarsening is simply amalgamating a group of elements or, in other words, undoing a previous refinement. Previous experience with h-refinement for graphics indicated that h-coarsening would follow quite naturally. For this reason it was decided to implement h-refinement and h-coarsening. P-refinement and p-coarsening, can be addressed at the element level, and could therefore be added at a later date if desired.

Manufacturing processes are frequently 3-D, nonlinear and transient. It is in these analysis that the problem of solving large matrices becomes critical. However, a review of the important literature revealed that previous adaptive algorithms have been developed primarily for 2-D finite element analysis. Unfortunately, extending mesh-management algorithms for 2-D analysis to those for 3-D analysis is not a trivial procedure. Quite simply stated, dealing with combinations of 4 edges is much easier than dealing with combinations of 12 edges and 6 surfaces. The most useful related research for 3-D analysis appears to be the study of 3-D mesh generation by Yerry,^{[83][84]} Cavendish^[85] and Shepard^[86]. Shepard^[86] has extended the mesh generation to include additional passes for localized refinement using tetrahedrons. However, several key issues were identified in the literature relating to 2-D adaptive analysis. These should be addressed before presenting the detailed formulation of the new algorithm. The issues involve h-refinement and the data structures which support adaptive management for a dynamic mesh.

3.1.1 Adaptive H-Refinement and H-Coarsening

During the development of the special routines to implement graphics for the grading elements, it became clear that h-refinement of these hexahedrons could be carried out conveniently and efficiently. It was appropriate to take advantage of any applicable advances made, by other researchers, for 2-D adaptive algorithms which deal with triangular and quadrilateral elements.^{[1][2][4][5][27][52][57]-(67)[87][88]}

Perhaps the most important contribution is that of Bank^[80] who has shown that bisection-type h-refinement is required to ensure the shape regularity of the elements created in the refinement process. Bisection-type refinement refers to the creation of new

elements (children) by the bisection of the sides of the parent element. A shape regular element is one whose side lengths are bounded above and below by fixed multiples of the element's characteristic length; e.g., $\sqrt{area}$, and all interior angles are bounded above 0 and below π .

Quadrilaterals are of more interest than triangles since they bear a closer relationship to hexahedrons, which are the subject of this research. In the case of quadrilaterals, Bank^[60] forms the children by joining the midpoints of opposite sides. He uses the error measures of Babuška^[68] and two primary rules to govern the refinement. First, any quadrilateral element which has more than 1 irregular node on any side must be refined (the 1-irregular rule). In combination with the 1-irregular rule, bisection-type refinement leads directly to binary; i.e., 2:1, grading. In binary grading no element edge equals the union of more than 2 adjacent element edges. Binary grading ensures there is a smooth transition of the function space between adjacent elements. Bank's^[60] second rule is that if an element is surrounded or nearly surrounded by refined elements then the element must be refined. In the case of a quadrilateral element, this will happen if there are refined elements are present on 3 or more sides (the $k_4 - 1$ rule). This appears to be done so the number of constraint equations is minimized. As refinement proceeds, the number of irregular nodes is reduced by also refining neighbouring elements so no quadrilateral has more than 1 irregular node on a side. Although Bank^[60] does not do so, this can be referred to as propagation of refinement. To limit the number of irregular nodes, additional refinement is needed in neighbours. Although it is not explicitly stated, Shepard^[66] also uses binary grading and propagation of refinement in his 3-D mesh generation.

The FEARS (finite element adaptive refinement system)-project of Babuška^{[19][46][67][68]} also uses linear quadrilateral elements for adaptive mesh refinement. Again, a bisection-type refinement is used. However, the number of irregular nodes is not necessarily limited. Nonconforming or irregular nodes are constrained to the value interpolated from neighbouring nodes. These irregular nodes require special tagging and complicate the calculation of the element level stiffness matrices. Irregular nodes can be avoided more easily when triangular elements are used. With triangles, both Bank^{[67][68][69]} and Rivera^{[61][62][63]} avoid irregular nodes by propagating refinement into neighbouring elements. Since the transition from large to small triangles is not abrupt, the triangulation is said to be smooth.

This brief review of refinement techniques has made three concepts clear. First, bisection-type refinement is a requirement for shape regularity. Second, binary grading avoids abrupt transitions from coarse to fine regions. Abrupt transitions are undesirable because they can lead to inaccuracies in the solution if the coarse side is too coarse. They are inefficient if the fine side is too fine. Third, propagation of refinement is unavoidable if the number of irregular nodes is to be limited and if binary grading leading to smooth transitions is to be maintained.

The 8- to 26-node hexahedrons can now be examined in light of the above discussion which makes clear the importance of binary grading, propagation of refinement, and smooth transitions from coarse to fine regions. First, in the implementation of the graphics any 8- to 26-node hexahedron is divided into its subelements (2, 4 or 8). This is not appropriate for h-refinement because, in an originally cubic element, division into 2 or 4 elements creates noncubic elements. Division into 8 elements does not. In an originally noncubic element, division into 2 or 4 elements will exaggerate any distortion. In these simple terms, refining an element into 8 subelements is a conservative strategy. Furthermore it guarantees both shape regularity and binary grading in all 3 directions. Second, with the 8- to 26-node hexahedron no special treatment of irregular nodes is needed since the constraints are embedded in the basis functions. In an initially uniform mesh, refining 1 element simply adds new nodes to the neighbouring elements. In an initially graded mesh, refining one element will naturally initiate propagation of refinement into a neighbour if a vertex node in the element under consideration is also a midside node or midsurface node in the neighbour. Finally, the resulting transition from fine to coarse regions is naturally smooth. Clearly, the 8- to 26-node hexahedron is extremely well suited to adaptive h-refinement.

3.1.2 Data Structures

A sound adaptive strategy should incorporate refinement to improve accuracy and coarsening to reduce the computational complexity. Obviously, managing the changes in a dynamic 3-D meshes is not a trivial issue. As Ewing^[89] indicates, a fairly sophisticated data structure is needed to support a general adaptive capability. Arrays and k-D trees have been selected as appropriate data structures in order to keep the computational complexity of retrieving data reasonably small. The arrays and their associated calculations are written in FORTRAN. Holtz^[96] implemented the k-D tree^[43] in PASCAL as an abstract data

type. Interestingly, Bieterman^[90] also found that numerical calculations were most easily implemented in FORTRAN whereas tree structures were more appropriate for languages such as PASCAL and LISP.

The fundamental concepts relating to the data structures for 3-D dynamic mesh management should be identified before describing the data structures in detail. There appear to be three prime requirements. First, there must be a complete definition of the finite element mesh. Second, the relationships among element neighbours must be known. Finally, the relationship between any parent element and its children must also be clear. These requirements will be clarified in the material which follows.

Any finite element mesh is completely defined by 2 distinct types of data. First, the topology defines the connectivity of the mesh. Second, in combination with the topology, the co-ordinates define the geometry of the mesh. These distinct types of data must be represented. Although the topology will indicate how many nodes are present in any element, it would be inconvenient to determine this repeatedly. Therefore it is also important to have a data structure which deals with the number of nodes in any element at any time. These three ideas can be represented by Defining Data Structures.

Since element labels are likely to change as elements refine and coarsen, the mesh should be managed (adapted) through nodal information. Each node must always be associated with a unique point in space. For example, during refinement a new node may appear in more than one neighbour. The new node must have the same identity in all elements and there should be no need for exhaustive searching for its identity. Similarly, in coarsening, a parent element should be retrievable if an identifying node can be determined from its children. The need to determine nodal identities and to retrieve the parent-child hierarchy suggests that tree data structures are appropriate. These data structures are also critical to the adaptive algorithm and hence can be called Critical Tree Data Structures.

If a mesh, or an element in a mesh, is too coarse, it should be refined. H-refinement is performed to improve accuracy by increasing the number of degrees of freedom (and thus enriching the function space). Consider what will happen if a given element is refined. To satisfy compatibility it is clear that new nodes must be added to some, perhaps all, of the neighbours in the appropriate positions in those neighbours. In a graded mesh, division of a given element might cause a neighbour to become improperly formed; i.e., to have too

many nodes. For this reason it might be necessary to refine neighbouring elements first. Checks on the neighbouring elements could show that their neighbours must, in turn, be refined first. In other words, the refinement propagates. A knowledge of element neighbours is critical if this path of propagation is to be properly determined. Similarly, h-coarsening is performed to reduce the computational cost of the analysis by reducing the number of degrees of freedom. If there are fine elements in a mesh, and all of these elements have very low errors, then the possibility of amalgamating or coarsening elements exists. If a group of elements is to amalgamate or coarsen they must first know the neighbours in their group. After the group has become one new element some nodes may no longer be needed to define the geometry. These redundant nodes could be removed from the appropriate positions in the element and its neighbours. Clearly, for both refinement and coarsening knowing the relationships among neighbours is critical. It will be shown that 2-D arrays are appropriate data structures representing such relationships and can be called Critical Array Data Structures. These critical array data structures will also support the insertion and deletion of nodes as the mesh changes.

3.1.2.1 Defining Data Structures

The purpose of any adaptive mesh-management algorithm is to enhance the capability of its associated finite element program. The adaptive algorithm presented here will eventually be linked to the 3-D, nonlinear, transient finite element code ASGAR. If the data structures which represent the topology and co-ordinates in the finite element code are efficient, it would be natural to use similar data structures to represent the topology and co-ordinates for the adaptive algorithm. ASGAR uses array data structures to represent both the topology and co-ordinates. These will be duplicated in the adaptive algorithm. The names of these arrays are: LNODS, NCOORD and LNODZG. In a dynamic mesh the number of nodes and elements will constantly change. As a result these arrays require a large upper limit on their size. At this time a value of 2000 has been arbitrarily selected.

The topology of the elements and the connectivity of the mesh are stored in the array LNODS which is a fundamental array from which all neighbouring relationships can be determined.

LNODS[1:26,1:2000] holds, for each element in the mesh, the list of identities of its 8 to 26 nodes. Positions 1 to 8 hold vertex nodes, positions 9 to 20 hold midedge nodes and

positions 21 to 26 hold midsurface nodes. Absent nodes are indicated by a 0. A 0 in the first position will be used to indicate that the element has been flagged for removal due to refinement or coarsening.

The following procedures are used:

1. $\text{assign}(0, A')$: assign the value 0 to the array A' , where A' is the 2-D array LNODS.
2. $\text{put}(file, A')$: copy $file$ over A' , where $file$ holds LNODS for new mesh.
3. $\text{get}(v, A')$: retrieve a column vector v_h from the array $A'_{h,i}$, where $1 \leq h \leq 26$ describes the local positions of the nodes and i is the element of interest.
4. $\text{put}(v, A')$: copy a column vector v_h into $A'_{h,i}$, where $1 \leq h \leq 26$ describes the local positions of the nodes and i is an element with a new topology.
5. $\text{delete}(i, A')$: initialize $A'_{h,i}$ to zero, where $h = 1$ and i is the element to be removed.
6. $\text{write}(A', file)$: output A' to $file$, where A' is the updated LNODS which holds the new topology.

Co-ordinates which define the geometry of the mesh are stored in the array NCOORD. NCOORD is used with the finite element basis functions to determine co-ordinate keys for the tree data structures and for determining element centroids. In addition it is used to determine when a midedge or midsurface node is not centred between its respective vertex nodes.

NCOORD[1:3,1:2000] holds, for each node in the mesh, its global x,y,z co-ordinates.

The following procedures are used:

1. $\text{assign}(5.5E+25, C')$: assign the value $5.5E+25$ to the array C' , where C' is the 2-D array NCOORD.
2. $\text{put}(file, C')$: copy $file$ over C' , where $file$ holds NCOORD for new mesh.
3. $\text{get}(v, C')$: retrieve a column vector v_h from the array $C'_{h,i}$, where $1 \leq h \leq 3$ describes the x, y and z co-ordinates of node i .
4. $\text{put}(v, C')$: copy a column vector v_h into $C'_{h,i}$, where $1 \leq h \leq 3$, describes the x, y and z co-ordinates and i is the new node.
5. $\text{delete}(v, C')$: initialize a column vector v_h to $5.5E+25$ in $C'_{h,i}$, where $1 \leq h \leq 3$ describes the x, y and z co-ordinates and i is node to be removed.

6. `write(C', file)`: output C' , where C' is the updated NCOORD which holds the new co-ordinates.

Finally, because the 8- to 26-node hexahedrons are variable node elements, the vector LNODEZG is used to store the true number of nodes in each element. As the topology of each element is placed in LNODS, its number of nonzero nodes is counted and stored in LNODEZG. This number is used repeatedly in calls to routines which use loops over the actual number of nodes. Typically, these routines calculate the positions of new nodes. Of course, as nodes are added to an element, its number of nodes changes. Further calculations of new nodal co-ordinates for that element should use all the existing nodes in the element. LNODEZG[1:2000] holds, for each element in the mesh, its true number of nodes.

The following procedures are used:

1. `assign(0, D')`: assign the value 0 to the array D' , where D' is the 1-D array LNODEZG.
2. `put(V, D')`: assign V , the number of nodes in element i , to D'_i ; where D'_i is LNODEZG for the new mesh determined from the array A' which holds LNODS of the new mesh.
3. `get(V, D')`: retrieve a value V from D'_i , where i is the element of interest.
4. `put(V, D')`: copy a value V into D'_i , where V is the updated number of nodes in element i .

3.1.2.2 Critical Tree Data Structures

Typically, coarsening or amalgamation of elements will follow refinement. The relationship between any parent element and its children is unique and hierarchical. If this hierarchical relationship is exploited through special data structures, then coarsening will follow quite naturally over several generations. The use of some form of tree data structure in mesh generation and adaptive finite element analysis is common.^{[1][2][4][5][19][27][57]–[60][66][67][82]–[87][91][92][93]} The forms of tree structure used by various researchers exploit the tree to provide information relating the elements in the various levels of the tree. Since a tree structure reflects the refinement process, each node of the tree typically has a label which identifies its type and allows reconstruction of the element's nodal co-ordinates. Other associated data are also stored so the parent and its neighbours, as well as each child and its siblings can be addressed. Other researchers^{[29][31][32][33][34]} who do not use tree structures also stress the ability to find relationships for nodal coordinates, neighbours and children. The

development of new data structures was not within the scope of this research. It was hoped that existing data structures could be implemented as an abstract data type or function, or used in a straight-forward manner. Two candidate data structures, oct-trees and k-D trees, were available for implementation as abstract data types. An existing solids modeller PADL-2^{[2][48][95]} includes oct-tree data structures which could have been used.^{[8][19]} Second, in the course of implementing the special routines for graphics of the 3-D elements (Appendix III) it was found that Holts^[96] implementation of the k-D tree, as an abstract data type, was invaluable in limiting the cost of finding individual nodal identities.

To develop fully automatic FEA, Kela,^{[2][4]} and Shepard^{[1][6][86]} exploit quad-tree and oct-tree data structures to generate finite element meshes as does Shepard's colleague, Yerry.^{[83][84]} Bachmann^[87] also exploits quad-trees for adaptive mesh generation and refinement for 2-D meshes. For mesh generation based on a geometric definition of the object using quad-tree encoding, the object or points of interest are placed inside a rectilinear region. Conceptually, this root region is divided into quadrants (children), the children are then divided again if they are at least partially occupied. The process is repeated recursively within each partially occupied region until some predetermined end condition is satisfied. The finite element mesh is the union of all the leaves. The structure of the quad-tree allows the neighbours of an element to be retrieved efficiently. For example, a knowledge of the location or connectivity of shared edges is often required in the error estimation portion of an adaptive analysis.^[87] With error estimates guiding refinement, the tree structure can then be used to control subdivision of the finite elements. One of the most important aspects of this data structure is that it is spatially addressable. The recursive spatial definition can be thought of as a directly addressable hierarchical grid in which the data associated with each leaf can be retrieved through a key that is a geometric region. To access the spatial data for a certain leaf, it is a simple matter to use an explicit formula or hashing function to obtain the exact address or pointer of the leaf. In other words, spatial regions can be associated with stored data, and stored data can be associated with spatial regions, all without searching the tree. This provides spatial addressability in constant time; i.e., at a cost of $O(1)$.

The oct-tree available with PADL-2^[48] is a full oct-tree for which space must be set aside for all possible children for about six generations. This leads to excessive storage requirements especially since space is not divided optimally and many of the leaves of the

structure are likely to be empty.^[2]

As indicated, quad-trees and oct-trees divide space into quadrants or octants. Usually some leaves of the tree are empty. Binary and tertiary homogeneous k-D trees^[43] can offer a significant advantage over quad-trees and oct-trees when all spatial points are available beforehand. In such a case, k-D trees may divide space optimally so all leaves are occupied.^{[97][98]} Furthermore, in k-D trees, points can easily be added. As each point is placed in the tree, the space is divided at that point. For instance, in 2-D cases, the first point divides the space in the X direction, the second point divides space in the Y direction. If points are ordered and added sequentially in a spiral fashion starting at the centre of domain, a well-balanced tree can be obtained.

Several functions are supported by k-D trees. Typical k-D tree operations include the following:

1. PTNEW - creates an identity for the k-D tree and allocates memory;
2. PTDEL - deletes the entire k-D tree to conserve memory or to allow it to be overwritten;
3. PTGET - checks for the existence of a key and returns the associated data;
4. PTPUT - puts data into the tree at a specified key;
5. PTGET - retrieves data from the tree.

For example, given a set of global co-ordinates called a key, the k-D tree subroutine PTGET responds with a status. If the status is 0, then that key (co-ordinates) exists in the tree and its associated data is returned to the calling program. If the status is 1, then the key (co-ordinates) does not exist. Other operations are also available but they are not used in this application.

During adaptive refinement and coarsening, the addition and removal of nodes at certain co-ordinates may affect the topology of several neighbours. In the division of the mesh-grading elements for graphics purposes (Appendix III) nodes are added in a locally sequential fashion similar to that described above. Unfortunately there is no guarantee that a well-balanced tree will be produced. An effort is made to add the points in a logical sequence since there is no way of knowing all the points *a priori*. The k-D tree was implemented in PASCAL^[96] as an abstract data type, and subsequently linked to ASGARD for the graphics. It was decided to continue to exploit the k-D tree for determining nodal

identities based on a co-ordinate key. If only nodes created during refinement are stored in the k-D tree, the amount of required storage will be reduced. The cost of extracting nodal identities from the k-D tree data structure is $O(\log_2 n)$, where n is the number of finite element nodes in the tree. The k-D tree for determining nodal identities is called CORTBL.

CORTBL stores the nodal identification for all created nodes. In effect, CORTBL maps the nodal co-ordinates in global (x,y,z) space into global nodal numbers, or identities. It is the inverse of the array NCOORD, which maps the node number onto its global co-ordinates, except that CORTBL contains only those nodes created during refinement. As nodes are added to elements, their co-ordinates are calculated. These co-ordinates act as a key. The key is sent to the k-D tree. If a node exists at those co-ordinates, its identity is returned. For example, if the key is $(2.1, 1.5, 4.6)$ the nodal identity might be 42. If a node does not exist at those co-ordinates a new identity (current maximum node number + 1) is established and stored in the tree at the co-ordinate key.

The following procedures are used:

1. $\text{create}(t_c)$: create an identity for the tree t_c and allocate memory. t_c is the k-D tree CORTBL.
2. $\text{put}(\text{info}, \text{key}, t_c)$: place info in t_c where key is the global co-ordinates of the node and info is the associated nodal identity.
3. $\text{get}(\text{info}, \text{key}, t_c)$: retrieve info from t_c where key is the global co-ordinates and info is the associated nodal identity.

There was no difficulty in extending the k-D tree for use in supporting the hierarchical parent-child relationships. During refinement, the topology of the parent element can be stored for each child element at a key which is the child's centroid. Later, coarsening can be accomplished by using the child's centroid as a key to extract the parent's topology. Some traversing or searching of the tree is required, but because the access is based on keys in the co-ordinate space of the mesh, spatial addressability provides very efficient parent retrieval at a cost of $O(\log_2 n)$, where n is the number of centroids in the tree. This is also the number of elements which are children. The k-D tree for parent retrieval is called PARTBL.

PARTBL stores the identity of the parent in terms of its nodes. As children are created, the parent identity is stored using the global co-ordinates of the child's centroid as the key

to the k-D tree. In effect, PARTBL maps the centroid co-ordinates of the children in global (x,y,z) space into the parent identity. The parent identity is defined by its full topology; i.e., 27 nodes. These are the original nodes in the element, the nodes added to midedges and midsurfaces so children could be created, and the node added at the centroid of the parent element. The latter is the node common to all siblings.

The following procedures are used:

1. $\text{create}(t_p)$: create an identity for the tree t_p and allocate memory. t_p is the k-D tree PARTBL.
2. $\text{put}(\text{info}, \text{key}, t_p)$: place *info* in t_p , where *key* is the global co-ordinates of the centroid and *info* is the associated full parent topology.
3. $\text{get}(\text{info}, \text{key}, t_p)$: retrieve *info* from t_p , where *key* is the global co-ordinates of the element's (child's) centroid and *info* is associated full parent topology.

In order to have access to the k-D tree at all times, PTNEW for both CORTBL and PARTBL is called from the uppermost routine. The alternative approach is to save the tree, using PTSAVE, at the end of each routine which uses the tree. In this case, the former was selected for convenience. It was felt that PTNEW would eventually be called from the finite element routine.

To appreciate the efficiency of data retrieval from a k-D tree consider the computational complexity of the following examples. To retrieve one piece of information from a k-D tree when the number of items in the tree is $n = 4000$, requires about 12 operations. To retrieve one piece of information from a k-D tree when the number of items in the tree is $n = 17000000$, requires approximately 24 operations.

3.2.1.3 Critical Array Data Structures

The relationships among elements will change when h-refinement is carried out. For example, a parent will be divided into children which are siblings. The relationship between the parent and its neighbours, will be different from that of the children to its parent's neighbours (consider any suburban neighbourhood). Although the k-D tree supports the parent-child relationships among the elements, a preliminary examination showed that, in a graded mesh, the tree would not easily handle relationships between neighbours. This is because a search of the area surrounding an element of interest, using a given radius, will

detect elements within the radius that are not immediate neighbours.

The change in relationships among elements suggested that the adjacency concepts of graph theory might offer a possible solution. Wilson^[99] showed that computer representation of a graph must be tailored to fit the particular application. He states that linked-list structures, which can be stored in noncontiguous memory, are suitable if extra graph adjacencies are added or existing adjacencies are removed. FORTRAN requires that data be stored in contiguous memory and therefore lists cannot be implemented easily in FORTRAN^[100], the language in which ASGARD is written. Moreover, no abstract data types for lists were immediately available in languages such as PASCAL. For this reason, it was necessary to look for other suitable data structures to represent the relationships among neighbours.

A simple solution presented itself when it became clear the relationships among neighbours can be extracted from the nodes because neighbours must be joined at the vertex nodes. They may also be joined at midedges and midsurfaces, but it is the vertex nodes that provide the more fundamental relationship. The midedge and midsurface nodes are able to define the path over which propagation of refinement will occur. Each vertex node is connected to, at most, 8 elements. This suggested that array data structures based on node numbers and their associated elements would be suitable. A preliminary review of the cost of using arrays was promising. Quite simply, data extraction of a single item from an array can occur in constant time, $O(1)$, if the exact array indices are known. This made the use of arrays a very attractive proposition. This must be balanced against the cost of filling the arrays as the mesh changes. The cost of filling the arrays is $O(n)$, where n is the number of nodes.

No doubt there are many ways to represent the fundamental relationships. In this case, three arrays which separate the neighbouring relationships into three fundamental subjects were selected. These are:

1. elements related by vertex nodes, NAYBORS[1:8,1:2000];
2. elements related by midedge nodes, MIDEDEG[1:4,1:2000];
3. elements related by midsurface nodes, MIDSUR[1:2,1:2000].

In each case the first dimension of the array is the limit that 3-D topology places on the number of elements that may share vertices, edges and surfaces. It is possible, in some

circumstances, to exceed these limits when the elements are not rectilinear. It is unlikely that a mesh generator would create a situation of this type. However, if the algorithm encounters this situation it issues a warning and stops. The second dimension will change as the number of nodes in the mesh changes. It is very difficult to determine how many new nodes will be created because, in an initially graded mesh it is possible to propagate refinement into every element in the mesh. Eventually ASGARD will estimate the size of the second dimension. For the time being a fixed upper limit of 2000 is used.

The separation of the neighbouring information into these three subsets allows the determination of the salient information at appropriate stages throughout the algorithm. A brief discussion of each of these arrays will clarify this point.

NAYBORS[1:8,1:2000] holds, for each node in the mesh, up to 8 elements in which the node is a vertex node; e.g. NAYBORS[l,33], $1 \leq l \leq 8$, might be 1 6 7 15 0 0 0. This means node 33 is a vertex node in elements 1, 6, 7 and 15.

NAYBORS deals only with the relationships between vertex nodes and elements. If nodes are to be added or deleted in neighbours then they must be added or deleted between specific vertex nodes. Thus edges and faces can be matched on the basis of vertex nodes alone. NAYBORS permits such identification of edges which match in up to 3 neighbouring elements and of faces which match in up to 1 neighbouring element. Once the edges and/or faces are matched, supporting data structures can quickly determine the exact local position at which the node must be inserted in the neighbour. Consider the 2-D mesh shown in Figure 3.1. If element 2 is to be divided, nodes must be added at the appropriate local positions in elements 1 and 3. Those locations can be determined by matching the vertex nodes for the edge shared by elements 1 and 2 and the edge shared by elements 2 and 3. Obviously, in a 3-D case edges and surfaces of neighbouring elements must be matched.

In coarsening, NAYBORS fulfils an additional, but related role. Each parent element creates children who are siblings. These siblings must know one another. NAYBORS fulfils this role since the node which is common to all siblings will be associated with the 8 element labels of those siblings.

The following procedures are used:

1. assign(0, G'): assign the value 0 to the array G', where G' is the 2-D array NAYBORS.
2. put(A', G'): enter the element number i in G'_{l,m} for every member of the LNODS array

$A'_{h,i}$, where $1 \leq h \leq 8$. The index m is the node number in $A'_{h,i}$ and l is the first unoccupied position in column m .

3. $\text{get}(v, G')$: retrieve a column vector v_h from the array $G'_{h,i}$, where $1 \leq h \leq 8$. The vector v_h gives the elements for which node i is a vertex node.

There are two critical array data structures which are fundamental to propagation of refinement. These are Midedg and Midsur. To illustrate the use of Midedg and Midsur consider the 2-D mesh in Figure 3.2. Assume that element 1 is requesting refinement.

When element 1 is to be refined, its neighbour must be refined first. If element 1 were refined first, element 2 would be given too many nodes and be improperly formed. Figure 3.2 shows refinement which propagates from a physically smaller element to a physically larger element. This need not be the case. Figure 3.3 shows how propagation can occur from a physically larger element to a physically smaller element, when the larger element is refined. This situation is typical of wedges and divergent internal boundaries. Clearly the physical size of the elements cannot be used to determine the direction of propagation.

Again, consider Figure 3.2. The vertex nodes of element 1; i.e., $\text{LNODS}[i,1]$, $1 \leq i \leq 8$, can be checked in turn, to see if they have elements in Midedg (and Midsur in 3-D). If a vertex node in the element to be divided is a midedge or midsurface node in a neighbour, then the neighbour must be refined first. The vertex nodes of that neighbour are then checked for propagation into other neighbours. A supporting vector called ELdiv is filled as the path of propagation is determined. Clearly, refinement must then begin at the last element in the propagation path described by ELdiv.

There are two applications for Midedg and Midsur in coarsening. Once coarsening occurs there may be some nodes which are no longer needed to define the geometry. These redundant nodes can be removed from the appropriate positions in the element and its neighbours. However there are two cases in which these nodes are not redundant and therefore should not be removed. First, midedge and midsurface nodes in the proposed parent element which are also vertex nodes in the proposed parent's neighbours, are not redundant. Second, midedge and midsurface nodes which are not within some specified tolerance of their linearly interpolated co-ordinates, based on their respective vertex nodes, are also not redundant. These nodes can be said to be carrying significant curvature.

In summary Midedg and Midsur are used to identify the path of propagation of

a

b

Fig. 3.1 Refinement in two dimensions
a. Original mesh; b. Mesh after refinement of element 2

a

b

Fig. 3.2 Propagation of refinement from a small element to a large element

a. Original mesh; b. Mesh after propagation of refinement from element 1

a

b

Fig. 3.3 Propagation of refinement from a large element to a small element
a. Original mesh; b. Mesh after propagation of refinement from element 1

refinement and to assist in node removal during coarsening.

MIDEDG[1:4,1:2000] holds, for each node in the mesh, up to 4 elements in which the node is a midedge node.

The following procedures are used:

1. assign(0, H'): assign the value 0 to the array H' , where H' is the 2-D array MIDEDG.
2. put(A' , H'): enter the element number i in $H'_{l,m}$ for every member of the LNODS array $A'_{h,i}$ where $1 \leq h \leq 4$. The index m is the node number in $A'_{h,i}$ and l is the first unoccupied position in column m .
3. get(v , H'): retrieve a column vector v_h from the array $H'_{h,i}$, where $1 \leq h \leq 4$. The vector v_h gives the elements for which node i is a vertex node.

MIDSUR[1:2,1:2000] holds, for each node in the mesh, up to 2 elements in which the node is a midsurface node.

The following procedures are used:

1. assign(0, I'): assign the value 0 to the array I' , where I' is the 2-D array MIDSUR.
2. put(A' , I'): enter the element number i in $I'_{l,m}$ for every member of the LNODS array $A'_{h,i}$ where $1 \leq h \leq 2$. The index m is the node number in $A'_{h,i}$ and l is the first unoccupied position in column m .
3. get(v , I'): retrieve a column vector v_h from the array $I'_{h,i}$, where $1 \leq h \leq 2$. The vector v_h gives the elements for which node i is a vertex node.

The three critical arrays can be used with each other and with the topology array (LNODS) in several combinations. For example, if element 8 is to be refined, its vertex nodes are extracted from LNODS and its neighbours can then be determined for each vertex node using NAYBORS. In all cases the access from one array to the next is direct and no searching is required. NAYBORS, MIDEDG and MIDSUR, are filled from the topology array each time a new mesh is created. Consider node 1, which might be a vertex node in element 8. In a mesh with many elements, 8 must be placed in the first row of the array NAYBORS. As each element is entered, it must search to find the first empty position in the row. Thus, the cost of filling the arrays is $O(n)$ where n is the number of vertex nodes, midedge nodes or midsurface nodes depending on the array. Once filled, data extraction from these arrays can occur in constant time if the exact indices are known. For example,

if placed in a Do-Loop with a counter, $1 \leq i \leq 8$, NAYBORS(1,i) will yield 8 elements, each retrieved in constant time, $O(1)$. All 8 neighbours can be retrieved at a cost of $O(1)$.

3.1.2.4 Supporting Data Structures

Additional supporting data structures are typically direct access arrays and data blocks. For example, ELDIV is a vector describing the path of propagation of refinement. Data blocks are used for invariant relationships. Typically these relationships involve the local space of the element. For example, the first node number in each element has the same local co-ordinates $(\xi, \eta, \zeta) = (1, 1, 1)$. Similarly, each of the 12 edges can be defined by 2 specific local vertex node numbers. Surfaces can be completely defined by four local vertex node numbers. The topology of each of the 8 subelements can also be defined by the 8 local node numbers of its vertices. If local co-ordinates are known then global co-ordinates can be found through the isoparametric transformation using the basis functions. Once the global co-ordinates are known, the global identity of the node can be extracted from the tree. The supporting data structures have their own costs. However, they are used in a very local fashion and are relatively small in comparison to the global data structures discussed above.

3.2 Formulation

The decision to refine, to coarsen or to retain an element must ultimately be reduced to a Boolean decision. For convenience, at the adaptive level, simple integer error indicators will be used.

1. An indicator which is a positive number is associated with an element whose contribution to the relative error exceeds the average.
2. An indicator of 0 is associated with an element which has an acceptable contribution to the relative error.
3. A negative indicator is associated with elements which can be coarsened because their contribution to the relative error is sufficiently low.

Eventually these indicators will be calculated from the relative error estimates developed by Mashaie.^[37] Mashaie's^[37] error estimates are based on error bounds in the strain energy. The error estimates are determined from the differences between the statically and kinematically admissible stress fields.

Large error indicators, whether positive or negative, indicate that more generations of refinement (large positive numbers) or coarsening (large negative numbers) are needed. The use of simple indicators such as these makes the algorithm independent of the specific form of error estimate used. This is a natural and proper division since the decision to alter individual elements is quite separate from the procedures for altering the mesh. Since the mesh-management algorithm functions as an independent entity it could be linked to any finite element code or mesh generator.

The algorithm has been designed to receive a mesh (topology and co-ordinates) and a set of error indicators for each element in the time step under consideration. Eventually, it will be directly linked to ASGAR, most likely as a higher level guiding routine. The description of the adaptive algorithm is divided into several sections. First, the rules and procedures for refinement and coarsening are presented. These procedures are then combined and presented in the form of pseudo-code for the mesh-management algorithm. Finally, a detailed description of the adaptive algorithm is presented using the specific subroutines which have been developed.

3.3 Refinement

The domain, Ω , is a bounded 3-D region, initially decomposed into a number, N , of well-formed 8- to 26-node hexahedrons, $\hat{e}^i$, which define the root mesh, $\mathfrak{R}$.

$$\hat{e}^i \in \mathfrak{R} : \Omega = \bigcup_{i=1}^N \hat{e}^i \text{ and } \bigcap_{i=1}^N \hat{e}^i = \emptyset \text{ and } 1 \leq i \leq N \quad (3.1)$$

Adaptive refinement of the root finite element mesh, $\mathfrak{R}$, describing Ω will improve the accuracy of the finite element solution. However, simple rules must be introduced to manage the complexity of refinement and propagation for the elements in the domain. Rules are needed to reduce the number of possible variations that can occur in refinement and propagation. In all cases the rules were chosen to be conservative; specifically, computational accuracy is favoured. Thus, in some cases, the refinement rules may affect elements which do not require refinement. The refinement rules (R1 and R2) introduced here serve similar purposes to those of Bank^[60] and Jiang^[29].

Rule R1: Each time an element is refined one level, it is divided, uniformly, into 8 subelements or octants.

Rule 1 ensures shape regularity,^[60] binary grading and reduces the program complexity. Thus h-refinement of any element, $\hat{e}^i$, creates 8 new subelements, $\hat{e}_\lambda^i$, such that:

$$\hat{e}^i = \bigcup_{\lambda=1}^8 \hat{e}_\lambda^i \quad (3.2)$$

Consider a uniform mesh composed of twenty-seven, 8-node elements with 3 elements on each edge. Refinement of the centre element will create a total of 12 midedge nodes, n_e , and 6 midsurface nodes, n_s , in the 26 neighbouring elements.

Rule 2: Refinement propagates to neighbours when a vertex node, n_v , in the element under consideration, $\hat{e}^i$, is also a midedge node, n_e , or midsurface node, n_s , in a neighbour, $\hat{e}^m$.

Rule 2 ensures that all elements are well formed and that the transition from coarse to fine elements is smooth. Thus refinement by propagation will occur from $\hat{e}^i$:

$$\forall \hat{e}^m, m \neq i \text{ where } n_v \in N_v(\hat{e}^i) = n_v \in N_v(\hat{e}^m) \text{ or } n_e \in N_e(\hat{e}^i) \text{ or } n_s \in N_s(\hat{e}^m) \quad (3.3)$$

Propagation of refinement occurs under these conditions and may well extend into elements which are requesting coarsening. This is not a problem because refinement improves the accuracy. It is a simple matter to override the request to coarsen.

These relationships among elements are consistent with other related work. For instance, Preiss^[101] uses similar information about vertices, edges and faces in his approach to checking the topological consistency of regular, conforming finite element meshes. Bank^[60] and Jiang^[29] utilize data relating vertex nodes, edge nodes and neighbours.

3.3.1 Procedures for Refinement

The refinement algorithm, operating in three dimensions, performs a sequence of procedures for those elements which have an error indicator which is a positive integer. The controlling procedure of the refinement algorithm is `Refine_Element`.

In a graded mesh, refinement of an 8- to 26-node brick can introduce the need for propagation of refinement into neighbouring elements. Propagation of refinement was illustrated for a 2-D case in Figures 3.2 and 3.3. If a vertex node in the element to be refined is also a midedge node or a midsurface node in a neighbour, then the refinement must propagate into the neighbour. The neighbour may, in turn, initiate propagation in its neighbours. This sequence of elements is called the propagation path. Simply stated, checking must begin in

the neighbours of the element to be refined. It must then radiate outwards to neighbours of neighbours and so on, until no more elements requiring refinement are detected.

The procedure `Propagate_Refinement` uses a forward, recursive loop to determine the correct propagation path. First, vertex nodes of the element to be divided are identified from `LNODS` and are checked to see if they have elements in the `MIDEDG` and `MIDSUR` arrays. If they do, then propagation must occur into the elements connected to that node. If propagation into a neighbour is indicated, then that neighbour is similarly checked for propagation into its neighbours. A vector of elements, `ELDIV[1:2000]`, describing the propagation path is filled accordingly. Propagation in three directions, with branching, is possible. However `ELDIV` is a one-dimensional (vector) data structure.

Refinement is then carried out, element by element using a backwards loop over the propagation path described by the propagation vector `ELDIV`. The refinement is accomplished through three procedures. First, neighbouring elements which need new nodes are found through the procedure `Match_Edges_and_Surfaces`. Nodes are added in the neighbours using the procedure `Add_Required_Nodes` and finally, the element is refined using the procedure `Divide_Element`.

The neighbours of the element to be divided are found using its vertex nodes, taken from the `LNODS` array, and `NAYBORS` which holds the elements owned by each vertex node. The edges in the neighbours which match those of the element are identified in the procedure `Match_Edges_and_Surfaces`. A new midedge node may need to be added in up to three neighbours between matching edge nodes.

Matching the edges requires several nested do-loops. The outermost loop is over the 12 edges of the element to be divided. The next loop is over the 2 vertex nodes on the edge. Finally there is a loop over the 8 possible elements (`NAYBORS`) owned by each vertex. These last two fill a local supporting structure `EL[1:2,1:8]`. Briefly, the first row of the local supporting structure contains up to 8 elements owned by the first vertex node. The second row contains up to 8 elements owned by the second vertex node. It is then a simple matter to use two nested loops to compare each item in row one to each item in row two. At most three matches can be identified. When a match is made, the element number and the two vertex nodes are placed in another local supporting structure `ADNDE[1:36,1:3]`. The dimension 36 is needed to accommodate the 12 edges and the 3 possible neighbours

for each edge. The dimension 3 is needed for each neighbouring element number and the 2 vertex nodes on the edge.

A second type of edge matching is also performed. In some cases 2 midedge nodes in the element to be divided are 2 vertex nodes in a neighbour. Thus, in the process of dividing the element, a midedge node must be created between the 2 vertex nodes in the neighbour. The pairs of midedge nodes which may match pairs of vertex nodes can be called surface couples. The procedure `Match_Edges_and_Surfaces` matches the surface couples to the edges in the neighbours. It requires several nested loops. The outermost loop is over the 12 surface couples of the element to be divided. The next loop is over the 2 midedge nodes of the surface couple for that edge. Finally there is a loop over the 6 possible elements (NAYBORS) for which the midedge node in the element to be divided is also a vertex node in a neighbouring element. The last two fill a local supporting structure `EL1[1:2,1:6]`. Briefly, the first row contains up to 6 elements for which the first node of the surface couple is a vertex node. The second row contains the elements for which the second node of the surface couple is a vertex node. Two nested loops are then used to match the elements in row 1 to the elements in row 2. When a match is made the element number and the 2 vertex nodes are placed in a local supporting structure `ADND1[1:24,1:3]`. The dimension 24 is needed for the 12 pairs of midedge nodes which form the surface couples and the 2 possible neighbours for each surface couple. The dimension 3 is needed for each neighbouring element and the 2 vertex nodes on the edge in the neighbour.

The surfaces in the neighbours which match those in the element are also identified in the procedure `Match_Edges_and_Surfaces`. A new midsurface node may need to be added to a neighbour between the 4 vertex nodes on the matching surface.

Matching the surfaces also requires several nested Do-Loops. The outermost loop is over the 6 surfaces of the element to be divided. The next loop is over the 4 vertex nodes on the surface. Finally there is a loop over the 8 possible elements (NAYBORS) owned by each vertex. These last two fill a local supporting structure `EL2[1:4,1:8]`. Thus, the first row of the local supporting structure contains up to 8 element owned by the first vertex node. The second row contains up to 8 elements owned by the second vertex node and so on. It is then a simple matter to use nested loops to compare each item in row one to each item in rows two, three and four until the matching face is found. When the match is made the element number and the four vertex nodes are placed in another local supporting

structure ADND2[1:6,1:5].

The procedure `Add_Required_Nodes` then adds the nodes to the neighbours at the appropriate local positions. The local structures ADNDE, ADND1 and ADND2 are used with LNODS to find the local positions, in the neighbours, of the midedge and midsurface nodes. At that point, the neighbour's topology is checked to see if the node exists at that local position. If it does not exist, a new node is created, its global co-ordinates are found and it is placed in the k-D tree CORTBL, and added to the co-ordinate array NCOORD. The co-ordinates of the new node can be found by using the co-ordinates and basis functions for the existing nodes in the element. As new nodes are created they are added to the neighbours in the correct local position in LNODS. These nodes are not added to the element to be refined since it will become 8 new elements.

Once all the nodes have been added where needed, the parent element enters the procedure `Divide_Element`. Each local position which is not a vertex node is checked; i.e., nodes 9 to 27. (Node 27 is created at the centre of the parent element, it is the only node common to all the children). If the node does not exist in the element's topology, its global co-ordinates are found. These co-ordinates are checked in CORTBL. If the node has been created, its identity is returned and used. If not, the node is created and added to the k-D tree PARTBL and to the array of co-ordinates NCOORD. The topology LNODS of the children is created element by element. The new elements are given temporary labels, each one higher than the current largest element number. This helps to minimize the frontwidth of the problem because smaller elements have higher labels than do larger elements. The children are assigned an error indicator which is one less than that carried by the parent.

The procedure `Divide_Element` also preserves the hierarchical parent-child relationships. The full topology of the parent just before division; i.e., 27 nodes, is stored in the k-D tree PARTBL for each subelement when the subelement is created. These 27 nodes are the original nodes in the element, the nodes added to midedges and midsurfaces so children could be created, and the node added at the centroid of the parent element. The latter is the node common to all siblings. The centroid of each subelement is used as the key to the k-D tree PARTBL. Since there is no way of knowing which child will be the first to be examined for coarsening, the parent topology must be stored for each centroid. An alternative would be to create another data structure to indicate which child was associated with the parent information.

The final step in the procedure `Refine_Element` is to re-initialize and re-establish the critical arrays (NAYBORS, MIDEEDG, MIDSUR). This accommodates the changes in the mesh due to the refinement of the element. Initializing prevents old information from coming into play after changes have been made. This step is skipped for the last element in the propagation path since it is performed by the refinement algorithm before the next element enters the procedure `Refine_Element`.

The refinement algorithm re-initializes and re-establishes the critical arrays (NAYBORS, MIDEEDG, MIDSUR). In addition, the full list of the topology and the co-ordinate array are rewritten. In this way the elements are relabelled and the divided element no longer appears in LNODS. At this point all nodes, both old and new, are present in LNODS and NCOORD. LNODZG is also re-established when the new mesh is read in. The refinement algorithm continues until all error indicators have been reduced to 0. Therefore, several generations of refinement can be handled without difficulty.

3.4 Coarsening

To the author's knowledge, this is the first algorithm which performs coarsening for 3-D meshes. Since, this algorithm must also accommodate graded meshes some thought was given to concepts such as maintaining shape regularity. Like refinement, coarsening requires certain conservative rules (C1 to C7) to maintain the computational accuracy.

H-refinement of the 8- to 26-node hexahedron can be accommodated by bisection-type division in which each element is subdivided into its 8 octants. This produces a unique hierarchical relationship between the parent and its children. As indicated this ensures the shape regularity of the element and guarantees binary grading. It is natural then, to reverse this process for coarsening so shape regularity and binary grading are maintained at all times. Therefore, coarsening makes use of the unique parent-child relationship so children can be coalesced or amalgamated by retrieving the proper parent element.

Rule C1: If Ω is a bounded 3-D region which has been previously refined in several regions, then 8 subelements can be coarsened into their parent element as follows:

$$\bigcup_{k=1}^8 z_k = z^i \quad (3.4)$$

Rule C2: Refinement takes precedence over coarsening.

This rule is invoked because refinement improves accuracy, whereas coarsening reduces the computational cost. Refinement is performed out of necessity, coarsening is performed for economy. It is possible for refinement to propagate across a mesh, into an element which has adequate accuracy and could be coarsened. If coarsening were to occur first, then propagation of refinement might undo the coarsening. By refining first, some of the candidates for coarsening may be removed from consideration. To maintain accuracy it was decided that refinement should be given precedence. For this reason, all refinement is carried out before coarsening begins.

Rule C3: Coarsening beyond the root mesh is not permitted.

The unique parent-child relationship which is required for coarsening does not exist if any of the elements to be coarsened are members of the root mesh, $\mathcal{R}$. Some form of arbitrary coarsening or adoption of parents will be needed for coarsening beyond the root mesh. This will be discussed later.

Rule C4: Coarsening is not permitted to propagate.

Rule 4 is a corollary, or natural consequence, of C2. To clarify this point, consider the concept of propagation of refinement. In refinement, if a vertex node in the element to be divided is a midedge or a midsurface node in a neighbour, then refinement must propagate. There is a similar issue for coarsening, in which a group of siblings wishes to amalgamate. If any of them has a midedge or midsurface node which is a vertex node in a neighbour, the neighbouring group needs to be coarsened first. This can be called propagation of coarsening, and if permitted, could undo previously performed refinement. Consider trying to coarsen the children of the larger parent in Figure 3.3b. Obviously, propagation of coarsening is nonconservative. Thus, only one group of siblings can be amalgamated at one time. This prevents previously refined elements from being coarsened due to a propagation of coarsening effect. Therefore, if a group of siblings wishes to coarsen and any of them has a midedge or midsurface node which is a vertex node in a neighbour a blockage in the coarsening is created. Accordingly, all the elements must be checked several times during the coarsening loop, to see if the changes which have been made, have freed previously blocked groups of siblings which also wish to coarsen. (See also rule C5.)

Rule C5: All siblings must have exactly 8 nodes.

When a sibling has more than 8 nodes it means that propagation of coarsening is needed

to coarsen the given group of siblings (see rule C4). Alternatively, a midedge node might be required to properly represent the geometry. In either case, the siblings should not be coarsened.

Rule C6: All siblings must agree to be coarsened; i.e., each one must have a negative error indicator.

There can be no sibling rivalry. Thus, if any of the siblings has an error indicator of 0, the error indicators for all the siblings are set to 0. This prevents a sibling which has the desired accuracy from joining a group to be coarsened.

The parent is given an error indicator which is one closer to 0 than that of its child with the indicator closest to 0. Since the optimum error indicator is 0, coarsening should move the indicator towards 0. This too is a conservative strategy because it prevents elements from being coarsened unnecessarily and thus minimizes the numerical errors by reducing the number of coarsening steps requested.

Rule C7: Nonredundant nodes must be retained.

A retrieved parent element has 27 nodes. Obviously, the body-centred node and some of the midedge and midsurface nodes are not required to represent the geometry. The nodes which can be removed are called redundant nodes. There are two types of nonredundant nodes. First, any midedge node or midsurface node in the retrieved parent which is also a vertex node in a neighbour is nonredundant and must be retained. Second, if a midedge or midsurface node in the retrieved parent is not within some allowable deviation of the linear midpoint or midsurface, then the node is not redundant and cannot be removed. This conserves the shape and volume of the domain Ω . In this case the allowable deviation has been set to 5% of the length of an edge containing a midedge node. Similarly, the allowable deviation is 5% of the length of one of the diagonals on a face containing a midsurface node.

3.4.1 Procedures for Coarsening

The coarsening algorithm is used for elements which have an error indicator which is a negative integer. To avoid blockages, the procedure `Coarsen_Element` is used in a backwards loop over the elements. Blockages occur when one sibling has a midedge or midsurface node which is a vertex node in a neighbour. Quite simply, elements created last will tend to block elements created first. Thus, elements with higher labels should be coarsened first since they will tend to block those with lower labels. Checking in this

manner is quite efficient and was thought to be preferable to trying to identify the path of elements that block coarsening and checking for agreement to coarsen along the path.

The keystone in the retrieval of the parent is the k-D tree PARTBL. This tree is filled during refinement so the unique, hierarchical parent-child relationships are accessible during coarsening. The centroid of the element is used to retrieve the parent topology and the common node from the k-D tree PARTBL. The common node is then used to extract the siblings of the element in question from the neighbours array NAYBORS. Rules C5 and C6 are checked to ensure that siblings have only 8 nodes and that they all agree to be coarsened. If rule C6 is not satisfied, the error indicators for all the siblings are set to 0. This removes the need to check them again later. If rules C5 and C6 are satisfied then the decision to coarsen has been made. To reduce the frontwidth, the parent is given the element label of its child with the lowest element label. This is a sensible thing to do but it is not essential. The error indicator of the retrieved parent is incremented from that of its child with the indicator closest to 0. The other siblings are flagged for removal from the topology list by setting the first position in LNODS for each sibling to 0.

At this point, the topology of the parent is filled from the data retrieved from PARTBL. The new element is fully compatible with the mesh but it has 27 nodes. Some of these nodes are redundant; that is, they are not required to represent the geometry of the domain, and can be removed through the procedure Remove_Redundant_Nodes. The body-centred node of the retrieved parent is always redundant. Each edge and each surface of the retrieved parent is checked to see whether the midedge and midsurface nodes can be removed. Any midedge or midsurface node which is also a vertex node in a neighbour cannot be removed. In addition, if the distance from the linearly interpolated midpoint to the midedge node is greater than the allowable 5% deviation, the node is said to be carrying significant curvature and cannot be removed. Similarly, if the distance from the linearly interpolated midpoint of a surface to the midsurface node is greater than the allowable 5% deviation, the midsurface node cannot be removed because it is carrying significant curvature. Each eligible node is removed from the topology LNODS of appropriate neighbours and from the topology LNODS of the retrieved parent. The node is also removed from the co-ordinate array NCOORD.

The final step in the coarsening algorithm is to re-initialize and re-establish the critical arrays (NAYBORS, MDEDG, MIDSUR) to accommodate the changes in the mesh

resulting from coarsening of a group of siblings. Initializing prevents old information from coming into play after changes have been made. In addition, the full list of the topology is rewritten so the elements are relabelled and any children which have coarsened no longer appear. At this point only existing nodes are present in LNODS and NCOORD. LNODZG is also re-established the next time a new mesh is read.

The coarsening cycle is then repeated for the next element. Since groups of siblings can be blocked from coarsening, coarsening is complete when the counter reaches 1 and there have been no changes made to the mesh. This allows negative error indicators to exist at the completion of coarsening.

3.4.2 Implementation of an Adaptive Mesh-Management Algorithm

The adaptive algorithm combines the algorithms for refinement and coarsening which have been dealt with above. However, it is appropriate to consider the general form of an algorithm which combines these features. The mesh-management algorithm is assembled from a series of small algorithms each of which performs a specific function. As indicated, the two main features are refinement and coarsening. The algorithm for refinement includes the following:

1. a controlling procedure `Refine_Element`;
2. a procedure for propagation of refinement `Propagate_Refinement`;
3. a procedure for matching edges and surfaces `Match_Edges_and_Surfaces`;
4. a procedure for adding nodes to the neighbouring elements `Add_Required_Nodes`;
5. a procedure for dividing the element into its children `Divide_Element`.

The coarsening algorithm begins with the procedure `Coarsen_Element` in which the parent's topology is retrieved and the rules for coarsening are checked. The procedure `Remove_Redundant_Nodes` is then used to remove the redundant nodes from the retrieved parent.

The pseudo-code which illustrates the combination of these procedures is presented below. A more detailed look at the specific subroutines of the mesh-management algorithm follows the pseudo-code. In both cases the descriptions are fairly general and do not cover various enhancements present in the actual code. For example, it is often possible to terminate nested Do-Loops before reaching the limits of the nested loops.

An Adaptive Mesh-Management Algorithm

Algorithm: Adaptive_Mesh_Management

Create k-D trees CORTBL and PARTBL

Create LNODS, NCOORD and LNODZG

Determine initial maximum node number NODMAX and initial maximum element number NELZ

Create NAYBORS, MIEDG and MIDSUR

Begin loop over error indicators

Start refinement

For each new mesh

Do Label 1 for elements 1 to NELZ

If error indicator > 0 , then Refine_Element

Overwrite LNODS, NCOORD, LNODZG

Update NODMAX and NELZ

Overwrite NAYBORS, MIEDG and MIDSUR

Label 1: Continue

Endfor

End refinement

Start coarsening

For each new mesh

Do Label 2 for elements NELZ to 1

If error indicator < 0 , Coarsen_Element

If element has coarsened, CHANGE = .TRUE. then

Overwrite LNODS, NCOORD, LNODZG

Update NODMAX and NELZ

Overwrite NAYBORS, MIEDG and MIDSUR

Endif

Label 2: Continue

Endfor

If NEL = 1 and CHANGE = .FALSE. coarsening is complete

End coarsening

End Adaptive_Mesh_Management

Procedure: Refine_Element

Assign element to propagation path ELDIV

Do Label 1 for vertices LNODS(1:8,NEL) in the element

 If vertex node has elements in MIEDG or MIDSUR then

 Assign element to ELDIV

 Propagate_Refinement

 Else vertex node has no elements in MIEDG and MIDSUR and
propagation path has been defined in ELDIV

 Endif

Label 1: Continue

Do Label 2 backwards over propagation path in ELDIV (starting with last entry)

 Match_Edges_and_Surfaces

 Add_Required_Nodes

 Divide_Element

 Re-create NAYBORS, MIEDG and MIDSUR (skip for last element in path)

Label 2: Continue

End Refinement_Element

Procedure: Propagate_Refinement

Get neighbour from MIEDG and MIDSUR

Propagate refinement to it; i.e., place neighbour in ELDIV

 Do Label 1 for vertices LNODS(1:8,neighbour)

 If vertex node has elements in MIEDG or MIDSUR then

 Assign element to ELDIV

 Propagate_Refinement to next neighbour

 Endif

 Label 1: Continue

End Propagate_Refinement

Procedure: Match_Edges_and_Surfaces

Do Label 1 over 12 edges in the element

 Do Label 1 over 2 vertex nodes

 Find elements owned by vertex in NAYBORS

 Do Label 1 over up to 8 elements owned by vertex 1

 Do Label 1 over up to 8 elements owned by vertex 2

 Compare owned elements

 If elements (other than element to be divided) match then

Place element and vertex nodes in ADNDE

Edge may match in up to 3 neighbouring elements

Else no match, continue

Endif

Label 1: Continue

Do Label 2 over 12 surface couples in the element

Do Label 2 over 2 midedge nodes which may match vertex node in neighbour

Find elements owned by potential vertex in NAYBORS

Do Label 2 over up to 6 elements owned by node 1

Do Label 2 over up to 6 elements owned by node 2

Compare owned elements

If elements (other than element to be divided) match then

Place element and vertex nodes in ADND1

Surface couple may match edge in up to 2 neighbouring elements

Else no match, continue

Endif

Label 2: Continue

Do Label 3 over 6 surfaces in the element

Do Label 3 over 4 vertex nodes

Find elements owned by vertex in NAYBORS

Do Label 3 over up to 8 elements owned by vertex 1

Do Label 3 over up to 8 elements owned by vertex 2

Do Label 3 over up to 8 elements owned by vertex 3

Do Label 3 over up to 8 elements owned by vertex 4

Compare owned elements

If 4 elements (other than element to be divided) match then

Place element and vertex nodes in ADND2

Surface may match in 1 neighbouring element

Else no match, continue

Endif

Label 3: Continue

End Match_Edges_and_Surfaces

Procedure: Add_Required_Nodes

Do Label 1: over 12 edges

Do Label 1: over 2 vertex nodes on edge

Do Label 1: over 8 local vertex positions

Find local positions of vertex nodes in neighbours by matching LNODS to ADNDE

Label 1: Continue

Do Label 2: over 12 surface couples

Do Label 2: over 2 vertex nodes on edge

Do Label 2: over 8 local vertex positions

Find local positions of vertex nodes in neighbours by matching LNODS to ADND1

Label 2: Continue

Do Label 3 over 6 possible surfaces

Do Label 3 over 4 vertex nodes

Do Label 3 over 8 local vertex positions

Find local positions of vertex nodes in neighbours by matching LNODS to ADND2

Label 3: Continue

Do Label 4 over 36 possible element-to-neighbour edges which match

Calculate local position of node to be added to neighbour (node is added between vertex nodes)

If node does not exist in element at that position then

Calculate global co-ordinates of the node from local co-ordinates

Check CORTBL

If node previously created, nodal identity is returned

Else

Increment node number $NODE = NODMAX + 1$

Put NODE in CORTBL at key = global co-ordinates

EndIf

Update LNODS of neighbour at proper local position

Update LNODZG of neighbour

Update NCOORD if required

EndIf

Label 4: Continue

Do Label 5 over 24 possible surface couple-to-neighbour edges which match

Calculate local position of node to be added to neighbour (node is added between vertex nodes)

If node does not exist in element at that position then

Calculate global co-ordinates of the node from local co-ordinates

Check CORTBL

If node previously created, nodal identity is returned


```

      Else
        Increment node number  $NODE = NODMAX + 1$ 
        Put  $NODE$  in CORTBL at key = global co-ordinates
      Endif
      Update LNODS of neighbour at proper local position
      Update LNODZG of neighbour
      Update NCOORD if required
    Endif
  Label 5: Continue
  Do Label 6 over 6 possible element-to-neighbour edges which match
  Calculate local position of node to be added to neighbour (node is centred between vertex nodes)
    If node does not exist in element at that position then
      Calculate global co-ordinates of the node from local co-ordinates
      Check CORTBL
        If node previously created, nodal identity is returned
        Else
          Increment node number  $NODE = NODMAX + 1$ 
          Put  $NODE$  in CORTBL at key = global co-ordinates
        Endif
        Update LNODS of neighbour at proper local position
        Update LNODZG of neighbour
        Update NCOORD if needed
      Endif
    Endif
  Label 6: Continue
End Add_Required_Nodes

```

Procedure: Divide_Element

```

For element to be divided
Place nodal identities for LNODS(1:8,NEL) in NUNDE. NUNDE will hold full parent topology.
Do Label 1 over local positions 9 to 27
  If node is present at local position, Add to NUNDE
  Else
    Calculate global co-ordinates from local co-ordinates
    Check CORTBL
      If node previously created, nodal identity returned
      Add to NUNDE at proper local position
    Else
       $NODE = NODMAX + 1$ 
    Endif
  Endif
End Do

```

```

      Put NODE in CORTBL at key = global co-ordinates
    Endif
  Endif
Label 1: Continue
Set LNODS(1,NEL) = 0 to flag removal of parent element
Current largest element, NEL = NELZ
Do Label 2 over 8 subelements
  NEL = NEL + 1
  If error indicator of parent > 0, decrement indicator for subelement
  Else error indicator of subelement = 0 (division occurred due to propagation)
  Endif
  Put NUNDE into LNODS(1:8,NEL)
  Set LNODZG(NEL) = 8
  Calculate global co-ordinates of centroid from local co-ordinates
  Put full parent topology NUNDE in PARTBL at key = global co-ordinates
Label 2: Continue
Endfor
Current largest element NELZ = NELZ + 8
End Divide_Element

```

Procedure: Coarsen_Element

For element to be coarsened

Calculate global co-ordinates of centroid

Check PARTBL

If parent does not exist Return. Attempt to coarsen beyond the root mesh is not permitted.

Else, parent exists

Retrieve full parent topology NODLST(1:27)

Common node of siblings = NODLST(27)

Use NAYBORS(common node) to find siblings

If any sibling has an error indicator $\geq$ to 0 there

is sibling rivalry, reset all sibling error indicators = 0, Return

Else all siblings have error indicators < 0 then

If any siblings has more than 8 nodes then coarsening is blocked, Return

Else Begin coarsening

Set NEL of parent = minimum label of siblings

Increment error of parent from that of children

Flag other siblings for removal LNODS(1,siblings) = 0

Put NODLST(1:26) topology into LNODS(1:26,NEL)

Remove_Redundant_Nodes

Endif

Endif

Endif

Endfor

End Coarsen_Element

Procedure: Remove_Redundant_Nodes

Do Label 1 over 12 edges of NEL

Use nodal identity of midedge nodes for edge, LNODS(9:20,NEL)

If midedge node has elements in NAYBORS then it is also a vertex node and is therefore nonredundant.
The node cannot be removed.

Else

Interpolate to find midpoint of the edge using NCOORD of vertex nodes

Calculate the length of the edge using NCOORD of vertex nodes

Find distance between actual co-ordinates of the midedge node and
the interpolated midpoint, divide by the length of the edge

If the distance/length is > .06

node is holding curvature and cannot be removed,

REMOV(local position) = .FALSE.

Else REMOV(local position) = .TRUE.

Endif

Endif

Do Label 1 over local positions I = 9 to 20

If REMOV(local position) = .TRUE. then

Find the local position J of this node in up to 3 neighbours
by comparing LNODS(9:20,NEL) to LNODS(9:20,neighbour)

Remove node from LNODS of neighbours, Reset LNODZG

LNODS(J,neighbour) = 0

LNODZG(neighbour) = LNODZG(neighbour) - 1

Remove node from NCOORD by setting to large number

NCOORD(1:3, LNODS(I,NEL)) = 5.5E+25

Remove node from parent element, Reset LNODZG(NEL)

LNODS(I,NEL) = 0

LNODZG(NEL) = LNODZG(NEL) - 1

Endif

Label 1: Continue

Do Label 2 over 6 surfaces of NEL

Use nodal identity of midsurfaces for surface, LNODS(21:26,NEL)

If the midsurface node has elements in NAYBORS then it is also a vertex node and is therefore nonredundant.
The node cannot be removed.

Else

Interpolate to find midpoint of the surface using NCOORD of vertex nodes

Calculate the length of the edge using NCOORD of vertex nodes

Find distance between actual co-ordinates of the midsurface node and
the interpolated midpoint and divide by the length of a surface diagonal

If the distance divided by length is > .05

node is holding curvature and cannot be removed,

REMOV(local position) = .FALSE.

Else REMOV(local position) = .TRUE.

Endif

Endif

Do Label 2 over local positions I = 21 to 26

If REMOV(local position) = .TRUE. then

Find the local position J of this node as a midsurface node in 1 neighbour
by comparing LNODS(21:26,NEL) to LNODS(21:26,neighbour)

Remove node from LNODS of neighbours, Reset LNODZG

LNODS(J,neighbour) = 0

LNODZG(neighbour) = LNODZG(neighbour) - 1

Find the local position J of this node as a midedge node in up to 2 neighbours
by comparing LNODS(21:26,NEL) to LNODS(9:20,neighbour)

Remove node from LNODS of neighbours, Reset LNODZG

LNODS(J,neighbour) = 0

LNODZG(neighbour) = LNODZG(neighbour) - 1

Remove node from NCOORD by setting to large number

NCOORD(1:3,LNODS(LNEL)) = 5.5E+25

Remove node from parent element, Reset LNODZG(NEL)

LNODS(LNEL) = 0

LNODZG(NEL) = LNODZG(NEL) - 1

Endif

Label 2: Continue

Remove common node of 8 siblings from NCOORD

NCOORD(1:3,common node) = 5.5E+25

End Removing_Redundant_Nodes

3.4.3 Subroutines for the Adaptive Algorithm

The current implementation of the algorithm is limited to problems in which the nodal co-ordinates and centroid positions do not change significantly. Transient heat transfer analysis and small deformation, small strain, stress analysis satisfy this condition. The extension to large deformation, large strain problems is somewhat more complicated and will be considered later.

Although the algorithm is reasonably compact and efficient, the main routines comprise about 4000 lines of code and comments. Space does not permit this code for the mesh-management algorithm to be included here. Therefore, detailed descriptions of the various routines for the algorithm are presented Appendix IV. As seen in the pseudo-code, the adaptive algorithm is a simple series of nested loops which manage the changes in the mesh. Figure 3.4 presents a chart of the structure of the algorithm. The major nested loops are shown in relation to the subroutine calls. Boxes are used to denote the major loops. Other less important loops are, of course, embedded in the code.

At the moment the adaptive algorithm is called by a small piece of code (MOKFEM) which mimics, in a very limited sense, a finite element solution. This code was written to establish a framework between a finite element algorithm and the adaptive routines. It controls the outermost loop (1) over the increments and opens the k-D trees CORTBL and PARTBL so they can be maintained in a multi-increment problem. The mesh is read by RDMSH and the error indicators are established by ERRCAL1. Alternatively, the error indicators can be set manually. The only things passed between MOKFEM and the adaptive code are the k-D tree identities, the increment number, the current maximum node number and the file names for the mesh and the error indicators. In a multi-increment problem, file names for the error indicators and new meshes are generated automatically.

At the adaptive level, the outermost loop (2) over the error indicators occurs in the routine CNTRL. This loop continues until the refinement error indicators have been reduced to 0, and the coarsening has been carried out as far as possible. At the top of this loop the critical arrays are initialized (INITLZ), the current mesh is read (RDMSH), the error indicators are read (ERRMSH), and the critical arrays are established (CONNECT). Within the loop over error indicators, there is a loop (3) over all the elements in the mesh to find those needing refinement. For each element requiring refinement, the routine RPNMSH is

-PTNEW(PARTBL)

-RDMSH

-ERRCAL

-CNTRL

1. Loop over increments.
2. Loop over error indicators.
3. Loop over elements for refinement.
4. forward recursive loop defines propagation path.
5. Backwards loop over propagation path.
6. Loop over edges, surface couples and surfaces adding nodes to neighbours.
7. Loop over local positions 9 to 27 adding nodes to parent.
8. Loop over 8 children.
9. Backwards loop over elements for coarsening.
10. Loop over edges and surfaces removing redundant nodes.

Fig. 3.4 Structure of the adaptive mesh-management algorithm

called. The refinement routine (RFNMSH) has a forward recursive loop (4) over neighbours of a given element to determine the path of propagation of refinement. This is followed by a backwards loop (5) over elements in the propagation path. The backwards loop has inner loops (6) over 12 edges, 12 surface couples and 6 surfaces. These inner loops determine when a neighbour needs new nodes before the element in the path is, itself, divided. When a neighbour requires a new node, the local position of the node is found for the neighbour. The topology of the neighbour is compacted (COMPACT) and the topology altering routine (TOPOL) is called. ADVAR3D evaluates the basis functions at the local position of the node to be added. POINTS3D then finds the global co-ordinates of the node. The k-D tree (CORTBL) is used to see if a node with these co-ordinates already exists. If so, its identity is returned. If not, the node is added to CORTBL. In either case, the new node is then added, in the correct local position, to the neighbour's topology by TOPOL. Once all neighbours have the required new nodes, the division routine DVIDE3D is called.

The division routine compacts the topology for the element to be divided (COMPACT). It then loops (7) over the potential sites (local positions 9 to 27) for new nodes. At each position, the topology altering routine (TOPOL) is called. It, in turn, calls ADVAR3D to evaluate the basis functions at each local position. POINTS3D is then called. It finds the global co-ordinates of the position. Finally, the k-D tree CORTBL is used to see if the node exists. At first glance it would appear that it is not necessary to call both PTGET and PTPUT since all new nodes are likely to have been previously created for the neighbours. However, this will not be the case for elements at exterior surfaces. Thus, additional new nodes may well be required. As the nodes are identified the full topology of this parent element is established in TOPOL. It is then a simple matter to loop (8) over the 8 subelements of the parent, creating the topology for the 8 children. As each child is created, ADVAR3D evaluates the basis functions at the child's centroid and POINTS3D converts the local co-ordinates of the centroid to global co-ordinates. The centroid is then used as the key to store the full topology of the parent in the k-D tree PARTBL.

It is possible for refinement to propagate throughout the entire mesh causing all the old elements to divide. Therefore, after the division of the element under consideration, RFNMSH must initialize and re-establish the critical arrays. This step is skipped for the last element in the backwards loop over the propagation path because it will be done by INITLZ after the return to CNTRL and after overwriting the altered topology and co-

ordinates (WRTMSH). Note that elements which have been divided are not considered in the re-establishment of the critical arrays.

The loop over elements for refinement is repeated until all refinement indicators have been reduced to zero.

Once the refinement has finished, there is a backwards loop (9) over elements to find those needing coarsening. This occurs within the loop (2) over error indicators. The coarsening loop over elements begins at NELZ and continues to 1. Therefore, the algorithm moves through the entire list of elements to be coarsened, coarsening where it can. Since each parent gets the label of its lowest labelled child, the loop continues one element further on the next time through.

The coarsening routine (CSNMSH) is called by CNTRL. Again, the topology of the element is compacted (COMPACT). The global co-ordinates of its centroid are determined by CNTRD. The basis functions for the local co-ordinates of the centroid (ADVAR3D) are evaluated. The local co-ordinates are converted to global co-ordinates (POINTS3D). Using the global co-ordinates as a key to PARTBL, the full topology of the parent is retrieved. Since the full topology includes the node which was common to all siblings, it is simple to retrieve each sibling's label from NAYBORS. When a group of siblings satisfies the conditions for coarsening there must be further loops over the 12 edges and 6 surfaces of the parent to remove the redundant nodes. Redundant nodes are removed from the topology of the parent and its neighbours and from the co-ordinates array NCOORD in loops (10) over edges and surfaces.

Finally, the topology and co-ordinates of the mesh are overwritten by WRTMSH, each time there is a return to CNTRL from CSNMSH. Coarsening is complete when the counter reaches the end of the loop over elements and there have been no changes in the mesh.

3.4.4 A Mesh-Management with Arbitrary Coarsening

It is important to note that it was possible to develop two techniques for coarsening. The first, discussed above, relies on the unique hierarchical parent-child relationships. The second, arbitrary coarsening, does not rely on these unique relationships.

Arbitrary coarsening can be thought of as a form of adoption since no unique relationship exists between the children and their parent. Instead children group together and coalesce to an undefined parent. Arbitrary coarsening has been developed primarily to

introduce the capability for substructuring and to provide an aid for mesh generation of graded meshes. For example, Downey^[103] has incorporated the arbitrary coarsening code in Ingrid.^[47] The uniform meshes generated by Ingrid can now be turned into graded meshes by coarsening appropriate areas.

Substructuring will require coarsening beyond the root mesh, R , to create an element from which the internal degrees of freedom have been removed.^{[103][104]} The data requirements for the substructure are different from those of an element. A stiffness matrix, boundary nodes and nodal reactions are typically associated with a substructure.

Eventually, the arbitrary coarsening will be linked to a substructuring routine. At the moment though, the arbitrary coarsening responds to a request to coarsen in the vicinity of a certain node which is common to some number of elements.

Rule AC1: In arbitrary coarsening, only groups of 2, 4 or 8 elements are permitted to join to form a new parent.

This rule prevents other combinations from forming wildly distorted elements. As in rule C4 above the children involved in forming the parent must have only 8 nodes.

$$\hat{e}^i = \bigcup_{h=1}^2 \hat{e}_h^i \text{ or } \hat{e}^i = \bigcup_{h=1}^4 \hat{e}_h^i \text{ or } \hat{e}^i = \bigcup_{h=1}^8 \hat{e}_h^i \quad (3.5)$$

The arbitrary coarsening procedure replaces the refinement procedure for unique relationships. The common node, selected by the user, is used with the array NAYBORS to determine the number of siblings associated with the common node. As indicated, the number of siblings must be 2, 4 or 8. The adoptive parent's topology LNODS, is directly related to the number of siblings and their orientation. Groups of 2 elements can be oriented in any of 3 directions, groups of 4 elements can be oriented in any of 3 directions. There is only one possibility for a group of 8 elements.

The centroid of each child is calculated and is used as the key to the k-D tree PARTBL which will hold the adoptive parent's topology. The full topology of the parent; i.e., 27 nodes, is stored in the k-D tree PARTBL for each child. In the case of adoptive parents which are groups of 2 or 4 elements, some of the midedge and midsurface nodes will not be present. It is important to note that in arbitrary coarsening, coarsening must be limited to one level at a time because adoption is used rather than a unique parent-child relationship. Therefore, the siblings are given indicators of -1 to trigger the regular coarsening routines

2

of/de

2

Micro-D

to begin coarsening. Once the parent elements have been adopted, the regular coarsening procedures are called to complete the alteration of the mesh.

3.4.5 Implementation of Arbitrary Coarsening

At the moment the implementation of arbitrary coarsening is limited to a mesh modification function. It has been found that root meshes are often too fine to permit cost effective analysis. In these cases the arbitrary coarsening routines have been very useful since time-consuming and tedious manual coarsening of elements can be avoided. However, the main reason for developing the arbitrary coarsening was for the eventual implementation of substructuring. The addition of substructuring to ASGARD is not within the scope of this research. Therefore, the additions for arbitrary coarsening will not be discussed in detail. Brief descriptions of the routines are included in Appendix V. At this time an interactive approach is used in a hunting routine so a region for coarsening can be selected on the basis of the chosen common node. The first adoption routine uses the common node to determine the number of possible siblings. The routine which creates the undefined parent is based on the number of siblings present; i.e., 2, 4 or 8.

3.5 Validation of the Mesh-Management Algorithm

The mesh-management algorithm has been used extensively for mesh generation.^{[40][42]} Three further examples of the capabilities of the adaptive mesh-management algorithm are presented below. In order to demonstrate the capability of the algorithm an arbitrary designation of the error indicators was used in each case. Refinement and propagation, refinement and coarsening, and adaptive management for a moving singularity problem, are illustrated. In the first two cases the error indicators were heuristically set by the analyst before using the the mesh-management algorithm. For the case of the moving singularity the error indicators were heuristically determined during the multi-increment problem by the code which mimics the finite element analysis. Arbitrary coarsening is not included because it functions in the same way as regular coarsening, although the definition of a parent is slightly different.

3.5.1 Refinement and Propagation

A case of extensive propagation of refinement is shown in Figure 3.5. Here, propagation extends to the left and right of element 3 which has an error indicator of 1. The recursive

forward loop identifies the elements that must be refined before element 3. The backwards loop over the propagation path attends to the division of the elements and the changes in the topology of the mesh.

The initial mesh has 30 nodes and 6 elements. After propagation and refinement the mesh has 94 nodes and 34 elements. The adaptive mesh management for this problem takes roughly 17 cpu seconds on an Apollo DN320.

3.5.2 Refinement and Coarsening

The hierarchical parent-child relationship for a 2-step problem is illustrated in Figure 3.6. Consider a case in which a point load is applied to a body in step 1 and removed in step 2. The user might select an initial mesh such as shown in Figure 3.6a which has 27 nodes and 8 elements. An examination of the errors during step 1, might indicate that 2 levels of refinement are needed in the element nearest the point load. Therefore the analyst sets the error indicator for that element to 2.

The first level of refinement of the parent produces children. The second level of refinement gives all the children, children of their own. In other words, 3 generations are present. In the refinement of the children, refinement also propagates into the neighbours as shown in Figure 3.6b. At this stage, there are 216 nodes and 113 elements in the mesh.

If the load were now removed, an examination of the errors during step 2, would likely indicate that the elements could be coarser. Therefore, the grandchildren coalesce to form children, redundant nodes are removed, and then the children coalesce to form the original parents. The final mesh is identical to the initial mesh.

Coarsening cannot propagate, therefore problems of this type can be accommodated in several ways, depending on the element numbering and thus the order in which the elements are considered in the coarsening routines. The adaptive mesh management for this problem takes about 100 cpu seconds on an Apollo DN320.

3.5.3 Adaptive Mesh Management for a Moving Singularity

Perhaps the most valuable application for the adaptive algorithm will be in the solution of time-dependent problems with moving singularities. To illustrate the capability of the algorithm to accommodate time-dependent situations, the problem of a point load moving along a bar was considered. For the purposes of illustration, the moving load can be

b

Figure 3.5 Refinement and propagation

a. Original mesh;

b. Mesh generated from request to refine element 3, showing effect of propagation

a

b

c

Figure 3.6 Refinement and Coarsening

- a. Original mesh; b. Mesh after 2 generations of refinement of element 1;
c. Mesh after 2 generations of coarsening

considered to be a moving heat source from a weld. The source moves along one edge of the bar, in a straight line.

This case also demonstrates that a functional framework has been established to connect a finite element analysis to the mesh-management algorithm.

The original mesh is read and error indicators are calculated and assigned to each element as would be done by a finite element analysis code. For testing purposes, the error indicators were based on the nondimensional ratio of V/R^3 . R is the distance from the heat source to the element's centroid and V is the volume of the element. The position of the source is calculated from its initial position, and the direction and speed at which it moves.

The elements undergo refinement in the vicinity of the moving load, and coarsen after the load has passed. The heat source travels the length of the bar in 6 time steps.

The original mesh is shown in Figure 3.7a. A refined area is introduced at the lefthand edge, and moved along the bar. A selection of the 6 finite element meshes generated by the adaptive algorithm is shown in Figures 3.7b to 3.7d. During the mesh management the mesh is constantly changing. The number of nodes varies between 110 and 215. Similarly, the number of elements varies between 40 and 96.

This problem clearly demonstrates the capability of the algorithm to deal with complex situations. This problem has similar features to the welding analysis shown in Figures 2.13 and 2.14 and was therefore run on an Apollo DN3000 so some feeling for the cost of the mesh management could be obtained. Mesh management for the entire 6 increment problem takes 292 cpu seconds (4.9 minutes) on an Apollo DN3000. This is a small fraction of the cost of 3-D large deformation, large strain, thermal-elasto-plastic stress analysis.

Fig. 3.7 A moving singularity

- a. Original mesh before calculation of error indicators; b. Mesh at the end of increment 2;
c. Mesh at the end of increment 3; d. Mesh at the end of increment 4

4. Automatic and Adaptive Finite Element Analysis

Fully automatic finite element analysis will be achieved when the adaptive mesh-management algorithm is linked to ASGARD. However, this linkage must include two other developments. They are: the relative errors of Mashaie^[37] and the data transfer of PROTEUS.^[48] A complete linkage of the adaptive algorithm to ASGARD is beyond the scope of this thesis because these two areas of related research are incomplete. Mashaie's^[37] error measures for 3-D meshes cannot yet be used for a large class of general problems. Second, PROTEUS^[48] is not yet at the stage at which it can handle a dynamic mesh. In a dynamic mesh the data must be properly transferred between the incremental meshes. However, it appears as if the adaptive algorithm will be a guiding routine called very high in the finite element code.

In a probable scenario, the finite element analysis will be run for one iteration and will then return the relative errors, which will provide the direction in which the mesh should be altered. The adaptive algorithm could then determine a more appropriate mesh. PROTEUS could transfer the nodal data, and the finite element analysis would restart or continue in some form. It is possible that each increment, perhaps even each iteration, will be considered as a new problem. There is the additional possibility of some form of shifting the adapted mesh of one increment, forward, to the next increment as mentioned by Ewing.^[89] A general capability for data transfer is now in place using ASGARD and PROTEUS.

4.1 Data Transfer

The adaptive algorithm or, alternatively, the finite element program must have some facility for data transfer between meshes. Each new mesh must completely specify the fields needed to define the initial conditions for that step.^[106] The most common approach to this problem appears to be simple interpolation of the fields from the nodal data. This approach is used by Jiang.^[39] This strategy will be accurate for a new mesh obtained from the bisection-type refinement used in the mesh-management algorithm. However, when a mesh is coarsened, interpolation is not accurate and mass, energy and momentum may not be conserved. Some form of projection must be used particularly when the fields of interest are not smooth and contain spikes or humps and hollows. Fortunately, the error in the

projection can be minimized if a least squares projection of the field, such as Galerkin's method, is used.

A least squares approach to data transfer can be used for both refinement and coarsening. This is effective in 3-D analysis and can be generally formulated by finding the new field which minimizes the error between the two fields.^{[48][105][106]} The integral of the difference between the two fields, weighted with the basis functions of new mesh, is set to zero. In a heat transfer problem two temperature fields are considered.

$$\int_{\Omega} (\bar{T} - T) N_j d\Omega = 0 \quad (4.1)$$

where:

N_j are the basis function for the new mesh;

$\bar{T} = \bar{T}_i \bar{N}_i$ is the temperature field for the old mesh;

$T = T_k N_k$ is the new temperature field, T_k are the unknowns;

Ω is the domain.

Therefore:

$$\int_{\Omega} (\bar{T}) N_j d\Omega = \int_{\Omega} (T) N_j d\Omega \quad (4.2)$$

$$\int_{\Omega} (\bar{T}_i \bar{N}_i - T_k N_k) N_j d\Omega = 0 \quad (4.3)$$

$$\int_{\Omega} \bar{N}_i N_j \bar{T}_i d\Omega = \int_{\Omega} N_k N_j d\Omega T_k \quad (4.4)$$

$$B_i = K_{jk} T_k \quad (4.5)$$

B_i and K_{jk} are known; K_{jk} is symmetric and positive definite. Therefore the unknowns T_k can be found:

$$T_k = K_{jk}^{-1} B_j \quad (4.6)$$

If the basis functions, $\bar{N}_i$, are lumped, then K_{jk} has terms on the main diagonal only, and the cost of solving equation 4.6 is small.

In this formulation, $\bar{T}_i$ and T_k must be nodal values of the field. In heat transfer, temperatures are mapped between the meshes. In elasto-plastic stress analysis, strains and stresses as well as state variables such as effective plastic strain, which are usually measured at the Gauss points, must also be transferred between the meshes. Research for transferring such data is underway.^[48]

The current version of PROTEUS reads the graphics file produced by ASGARD for the first mesh. This file contains the nodal temperatures and displacements for the nodes in the subelements. ASGARD has been modified slightly so the stresses, strains and state variables known for the Gauss points are projected to the subelement nodes using Patel's^[7] least-squares projection algorithm. The nodal identities for the first mesh are stored in a k-D tree at co-ordinate keys. PROTEUS also reads the topology and co-ordinates for the second mesh. It then loops over the nodes in the second mesh and checks them, in turn, for a match in the k-D tree. If a match is found, all nodal data are transferred to the second mesh. If no match is found the nodal data are interpolated from the surrounding matching nodes. The data associated with the Gauss points are then projected from the nodes to the Gauss points.

4.2 Error Estimates

A priori error estimates or error bounds are rarely available for practical problems.^[67] Heuristic error measures are often inappropriate.^[67] Thus, there is a need for a *posteriori* estimates which use the information obtained during the solution process. Many of the methods for guiding adaptive refinement and coarsening rely on some form of a *posteriori* error estimates.^{[1][19][27][29][32][45][57][58][61][62][64][66][67][69][81][86]–[89][107][108][109]} Of these, Shepard^[86] presents what appears to be the only adaptive capability in three dimensions. However, this adaptive refinement is only usable for 3-D mesh generation. Thus, in Shepard's^[86] case the adaptive aspect might be more appropriately called re-refinement since it is limited to an interactive request to refine the elements connected to a given node. It is a heuristic adaptive approach used in mesh generation and it is applied before the finite element solution is attempted. Fully automatic FEA requires adaptive mesh-management algorithms which can alter the mesh during the analysis.

A posteriori estimates provide the feedback for adaptive control of the solution process. Bank,^{[57][58]} uses Babuška's^{[37][52][66][67]} error measures and makes a clear distinction between global estimates of error, which are used to assess the accuracy of the solution, and local error estimates which are used to decide whether or not to refine a given element. Bank^{[57][58]} showed that the element level errors contribute to the global error. However, in the event of error pollution, the element level error does not necessarily measure the error in the element alone. From another viewpoint, if one element is refined to reduce its error, it may have the effect of reducing errors in other elements as well. With this mind, it is now possible to refer to element level error measures.

It will be necessary to relate Mashaie's^[37] error measures to the error indicators used in the adaptive algorithm. Mashaie's^[37] error measures are based on error bounds in the strain energy and are determined from the difference between the statically and kinematically admissible stress fields. The following procedures are proposed as a link between the error indicators used in the mesh-management algorithm and Mashaie's^[37] error measures. First, adaptive mesh management will only occur when the solution is unacceptable. The solution is defined to be unacceptable if the global relative error exceeds some tolerance set by the analyst.

$$\epsilon_n > \Gamma \quad (4.7)$$

When the solution is not acceptable, each element is examined in turn, to see whether it should be refined, left alone, or alternatively, coarsened. Refinement is performed to improve the accuracy of the solution. It will occur for those elements for which the elemental contribution to the relative error exceeds the average contribution to the relative error.

$$\epsilon_e > \frac{\epsilon_n}{N} \quad (4.8)$$

An alternative is:

$$\epsilon_e > \frac{\Gamma}{N} \quad (4.9)$$

In an acceptable mesh, where the error is equally distributed over the domain, all elements should be acceptable, needing neither refinement nor coarsening. Thus, there cannot be a fixed criterion above which elements are refined, and below which elements are coarsened. Some range of acceptable elements is required. For this reason, coarsening, which

reduces the computational complexity, should only be performed for elements with very low contributions to the relative errors. Therefore, elements will be coarsened if:

$$\epsilon_e < \frac{F \times \Gamma}{N} \quad (4.10)$$

Although some of the available references for 1-D and 2-D analysis mention the possibility of coarsening, the discussions are very limited. The literature provides no information on a choice of coarsening factor or indeed, when to coarsen. In order to gain some experience, a coarsening factor of F will have to be arbitrarily selected. A value of $F = 0.4$ would be a suitable place to start the investigation.

4.3 Advances for Error Indicators

The current approach to error indicators is to reduce them to 0, by either h-refinement or h-coarsening. Therefore, an 8-node brick with an indicator of 2, becomes 64 elements at the second level of refinement, each with an indicator of 0. This is an adequate approach for the moment. However, in the early stages of linking the adaptive mesh-management algorithm to PROTEUS, it will be necessary to closely consider the behaviour of the mesh. First, only integer values of -1, 0, and 1 should be used so experience with error indicators and the range of acceptable elements and the coarsening factor can be gained. The next step will be to determine whether averaging the error indicators, in some fashion, improves the behaviour of the mesh.

Knight^[110] developed a Smalltalk¹ prototype to test various forms of averaging element level error indicators for refining 2-D graded meshes. The element size and error were combined so grading occurred near the highest error indicators. In a 3-D sense, this means that an 8-node brick with an indicator of 2, would not become 64 elements of equal size. Instead it would become 7 large elements at the first level of refinement surrounding 8 small elements at the second level of refinement.

Knight^[110] investigated six forms of averaging neighbouring element error indicators based on an initially uniform mesh. Rounding up and down, truncating, element size, as well as including and excluding negative error indicators, were considered. Several conclusions were made:

¹ Smalltalk is an object-oriented programming language.

1. Refinement should occur before averaging. Refinement indicators should then be reset based on the averaging of neighbours.
2. Only elements requesting refinement should be grouped together for averaging. This is because averaging elements which wish to be coarsened with those that wish to be refined will artificially lower the resulting indicator.
3. Rounding the averages up appears to produce the best graded mesh.

The study of averaging techniques should be extended to test these concepts on initially graded meshes. Although grading towards the highest indicator offers some promise, it is conservative to continue to divide elements uniformly at each generation. Consider for example, the test of the Serendipity analogue presented earlier in section 2.2.3. Although the subelement contributions to the relative error were not presented, they all had a larger contribution than the surrounding elements. In effect, all the subelements were poor, and all should be divided. Thus, uniform refinement of the element with the largest error indicator, rather than averaged and graded refinement is likely to be the preferred approach in the initial stages.

Studies similar to those above must be carried out for h-coarsening. However other approaches will be needed when the mesh-management algorithm is extended to p-refinement and p-coarsening. The concept which is common to both h-refinement and p-refinement is the desire to add those degrees of freedom which will have the largest contribution to the increase in global accuracy.^[81] In h-coarsening and p-coarsening, the related concept is to remove those degrees of freedom which do not contribute significantly to the global error. This emphasis on the degrees of freedom suggests that degree of freedom indicators are needed for refinement and coarsening rather than an elemental contribution to the relative error. Babuška^{[37][53][66][67]} has published extensively on this. Certainly, combined h- and p-refinement and combined h- and p-coarsening as well as degree of freedom error indicators should be considered as a subject for future research for the mesh-management algorithm.

4.4 Large Deformation

The first linkage of the adaptive algorithm to ASGARD will be limited to the analysis of problems in which the nodal positions do not change appreciably. Transient heat transfer analysis and small deformation, small strain, stress analysis satisfy this condition. The

extension to large deformation, large strain problems adds a certain degree of complexity to the algorithm.

In thermal-elasto-plastic analysis using data transfer, there is no reason why the heat transfer discretization and stress analysis discretization must be the same. The adaptive algorithm could produce the meshes which are most suitable for the two decoupled halves of a thermal-elasto-plastic analysis. In this way, the large deformations which often accompany thermal expansion, could be handled appropriately. However, in large deformation problems, nodes and element centroids move. This presents a difficulty in the use of co-ordinate keys for the k-D trees CORTBL and PARTBL. Since element labels change constantly, there is no sensible way in which element labels can be used to track changes in the mesh. The only variables available during the adaptive analysis are the nodal co-ordinates and the node numbers.

The nodal co-ordinates are updated in ASGAR during the large deformation analysis. It would be a simple matter to delete (PTDEL) and refill the co-ordinate k-D tree CORTBL when there has been large deformation. Some of the efficiency will be lost, because the k-D tree would be holding all nodes in existence, rather than only those created during refinement operations. New nodes would, of course, be added on subsequent refinements. If the same element were refined and coarsened several times, it would have a different node numbering each time, because new nodes are numbered incrementally. Again, although this would result in a loss of efficiency, the adaptive algorithm would have no difficulty coping with the numbering changes. Another possibility is to allow more flexibility in matching the key to the value in the tree. This flexibility is often called a fuzzy factor.

The real difficulty lies in the retrieval of parent topology from PARTBL. Here, the key to the tree is the centroid of a child element. Since the hierarchical parent-child relationships are unique and multigenerational problems must be considered, some way of updating the centroids must be developed. Unlike nodes, there are few existing finite element computations at the centroids. In thermal-elasto-plastic analysis, element centroids are sometimes used as integration points for the temperature field. However, this offers little assistance. A larger fuzzy factor may be one solution. Alternatively, Otoo's^[91] range finding algorithm may offer a solution. The range finding algorithm searches for a match within a given range of values. A third possibility is to treat the situation much like a total Lagrangian finite element analysis, in which all information is taken with respect to the

original geometry. Thus, a discrete connection between the old and new geometries could be used. This would also remove the need for the k-D tree CORTBL to hold all the nodes.

Finally, a form of index space could be used to capture the topology. INGRID^[47] uses index space to generate meshes. The object is placed so it is centred in an index space with limiting co-ordinates between $(x, y, z) : x, y, z \in \text{integers} > 0$. The topology of the elements is created and mapping functions move the element boundaries, etc. into the proper shape. There is a direct correlation between the real co-ordinates of a node, the element topology and the co-ordinates in index space. Therefore, no matter what the shape of an element, its nodes always have the same position in the index space. This index position could be used as the link between the original and final co-ordinates of the centroids and nodes which have moved.

4.5 Fully Automatic Finite Element Analysis

The adaptive mesh-management algorithm is an important development for automatic FEA. As indicated earlier, data transfer between meshes and rigorously generated error measures are also fundamental requirements. At this time it is not possible to link the adaptive algorithm to ASGARD. Unfortunately, fully automatic FEA cannot be demonstrated. Nevertheless it is clear that fully automatic finite element analysis will be realized in the near future. The adaptive mesh-management algorithm will be an integral part of this advanced technique.

5. Conclusions and Contributions

1. 2-D and 3-D linear mesh-grading elements have been proposed for finite element analysis of problems which contain localized phenomena. It has been shown that cost savings are significant for problems with a small number of elements and that in larger problems a graded mesh offers dramatic cost savings over its corresponding uniform mesh. Moreover, element distortion is minimized and subspace richness is maximized for these graded finite element meshes.
2. The 8- to 26-node hexahedron has provided an advance in mesh grading which has allowed a work station to perform 3-D thermal-elasto-plastic analysis faster than current supercomputers using traditional analysis. The computational complexity for an implicit analysis of a point load on a 3-D uniform mesh is $O(n^7)$. This has been reduced to $O(\log_2 n)$ for the equivalent recursively graded mesh. The 8- to 26-node hexahedron made it possible to perform the first fully 3-D thermal-elasto-plastic analysis of a weld.
3. The mesh-grading elements fit naturally with spatially addressable data structures. This has been exploited in the development of a rule-based adaptive mesh-management algorithm which creates dynamic and hierarchical 3-D finite element meshes. The adaptive mesh-management algorithm will form an integral part of fully automatic and adaptive finite element analysis.

6. References

- [1] M. S. Shepard, 'Adaptive Finite Element Analysis and CAD', *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, John Wiley and Sons Ltd., Chichester, U.K., 205-225, 1986.
- [2] A. Kela, H. Voelcker and J.A. Goldak, 'Automatic Generation of Hierarchical, Spatially Addressable Finite-Element Meshes from CSG Representations of Solids', *Int. Conf. on Accuracy Estimates and Adaptive Refinements in Finite Element Computations (ARFEC)*, Lisbon, Portugal, June 1984.
- [3] A. Kela, R. Perucchio and H. B. Voelcker, 'Toward Automatic Finite Element Analysis', *Computers in Mech. Engng*, 5, 57-71 (1986).
- [4] A. Kela, *Automatic Finite Element Mesh Generation and Self-Adaptive Incremental Analysis through Solid Modeling*, Ph.D. Thesis, University of Rochester, Rochester, N.Y., January 1987.
- [5] M. S. Shepard and K. H. Law, 'The Modified-Quadtree Mesh Generator and Efficient Adaptive Analysis', *Int. Conf. on Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, Lisbon, Portugal, June 1984.
- [6] A. P. Chakravarti, J. A. Goldak, and A. S. Rao, 'Thermal Analysis of Welds', *Proc. of Fourth Int. Conf. on Numerical Methods in Thermal Problems*, Swansea, UK, July 1985.
- [7] B. Patel, *Thermo-Elasto-Plastic Finite Element Analysis of Deformations and Residual Stresses due to Welds*, Ph.D. Thesis, Carleton University, Ottawa, January 1985.
- [8] J. Goldak, M. McDill, A. Oddy, M. J. Bibby, R. House and X. Chi, 'Computational Heat Transfer for Weld Mechanics', *International Trends in Welding Research Conference*, Gatlinburg Tennessee, May 1986.
- [9] J. Goldak, A. Oddy, M. McDill, M. J. Bibby, R. House and A. P. Chakravarti, 'Progress in Computing Residual Stress and Strain in Welds', *International Trends in Welding Research Conference*, Gatlinburg Tennessee, May 1986.
- [10] J. Goldak, M. McDill, A. Oddy, M. J. Bibby, R. House and A. Mashaie, '3-D Thermo-Elasto-Plastic Analysis of Arc Welds', *Symposium on The Computer Modelling of*

Fabrication Processes and Constitutive Behavior of Metals, Ottawa, Canada, May 1986.

- [11] J. A. Goldak, B. Patel, M. Bibby and J. Moore, 'Computational Weld Mechanics', *Invited Opening Paper for AGARD Workshop, Structures and Materials 61st Panel Meeting*, Oberammergau, Germany, September 1985.
- [12] J. A. Goldak, A. P. Chakravarti and M. J. Bibby, 'A New Finite Element Model for Welding Heat Sources', *Trans. AIME*, 15B, 299-305 (1984).
- [13] J. M. McDill, A. S. Oddy, J. A. Goldak, M. J. Bibby, 'Progress in Simulation and Optimization of Manufacturing Processes', *Proc. S.W.E. National Convention*, Kansas City, Mo., June 24-26, 1987.
- [14] J. C. Diaz, 'Time-Dependent Self-Adaptive Mesh Modification Techniques', *Proc. of Conf. on Mathematical and Computational Methods in Seismic Exploration and Reservoir Modeling*, Houston, Texas, January 21-24, 1985.
- [15] G. F. Carey and J. T. Oden, *Finite Elements, A Second Course Vol. II*, Prentice-Hall, Englewood Cliffs, N.J., 84, 1983.
- [16] K.-J. Bathe, *Finite Element Procedures in Engineering Analysis*, Prentice-Hall, Englewood Cliffs, N.J., 220-231, 1982.
- [17] A. S. Oddy, J. A. Goldak, J. M. McDill, M. J. Bibby, 'A Distortion Metric for Isoparametric Elements', accepted *CSME Transactions*, November 1987.
- [18] ARFEC, *Int. Conf. on Accuracy Estimates and Adaptive Refinements in Finite Element Computations (ARFEC)*, Lisbon, Portugal, June 1984.
- [19] W. C. Rheinboldt and C. K. Mesztenyi, 'On a Data Structure for Adaptive Finite Element Mesh Refinements', *ACM Trans. on Math. Software*, 6, 2, 166-187 (1980).
- [20] J. M. McDill, J. A. Goldak, A. S. Oddy and M. J. Bibby, 'Isoparametric Quadrilaterals and Hexahedrons for Mesh-Grading Algorithms', *Comm. Appl. Num. Meth.*, 3, 155-163 (1986).
- [21] J. C. Cavendish, 'Local Mesh Refinement Using Rectangular Blended Finite Elements', *J. Computational Phys.*, 17, 211-228 (1975).
- [22] J. C. Cavendish, W. J. Gordon, 'Substructured Macro Elements Based on Locally Blended Interpolation', *Int. j. numer. methods eng.*, 11, 1405-1421 (1977).
- [23] P. T. Walsh, 'Intensive Finite Element Grading for Stress Concentrations', *Engng Frac-*

- ture Mech, 10, 211-213 (1978).
- [24] K. K. Ang and S. Villoappan, 'Mesh Grading Technique Using Modified Isoparametric Shape Functions and Its Application to Wave Propagation Problems', *Int. j. numer. methods eng.*, 23, 331-348 (1986).
 - [25] A. K. Gupta, 'A Finite Element for Transition from a Fine to a Coarse Mesh', *Int. j. numer. methods. eng.*, 12, 35-45 (1978).
 - [26] I. J. Somervaille, 'A Technique for Mesh Grading Applied to Conforming Plate Bending Finite Elements', *Int. j. numer. methods eng.*, 6, 310-312 (1972).
 - [27] I. Babuška, 'Feedback, Adaptivity and *a posteriori* Estimates in Finite Elements: Aims, Theory, and Experience', *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, John Wiley and Sons Ltd., Chichester, U.K., 3-23, 1986.
 - [28] M. Fortin and P. Tanguy, 'A Non-Standard Mesh Refinement Procedure through Node Labelling', *Int. j. numer. methods eng.*, 20, 7, 1361-1365 (1984).
 - [29] B. N. Jiang and G. F. Carey, 'Adaptive Refinement for Least-Squares Finite Elements with Element-By-Element Conjugate Gradient Solution', *Int. j. numer. methods eng.*, 24, 569-580 (1986).
 - [30] O. C. Zienkiewicz, *The Finite Element Method*, 3rd Ed., McGraw Hill Book Co., London, 1979.
 - [31] G. F. Carey, 'A Mesh-Refinement Scheme for Finite Element Computations', *Comp. Meth. Appl. Mech. Eng.*, 7, 93-105 (1976).
 - [32] B.A. Szabo, 'Estimation and Control of Error Based on P- Convergence', *Int. Conf. on Accuracy Estimates and Adaptive Refinements in Finite Element Computations (ARFEC)*, Lisbon, Portugal, June 1984.
 - [33] K-J. Bathe, *Finite Element Procedures in Engineering Analysis*, Prentice-Hall, Englewood Cliffs, N.J., 275, 1982.
 - [34] D. Guildenhuis, Private Communication, Ottawa, Canada, November 1984.
 - [35] K-J. Bathe, *Finite Element Procedures in Engineering Analysis*, Prentice-Hall, Englewood Cliffs, N.J., 485, 1982.
 - [36] J. A. Goldak, *88.566 Nonlinear Finite Element Analysis*, Carleton University, Ottawa, Canada, 1983.

- [37] A. Mashaie, *Error Bounds for Adaptive Finite Element Analysis With an Application to Elastic Boundary Value Problems*, Ph.D. Thesis, Carleton University, Ottawa, Canada, in progress.
- [38] L. L. Meekisho, *Finite-Element Analysis of Contact and Dynamic Problems for Manufacturing Processes*, Ph.D. Thesis, Carleton University, Ottawa, Canada, in progress.
- [39] D. Stillman, *TAURUS, A Post-Processor for Finite Element Analysis*, Lawrence Livermore National Laboratory, California, 1981.
- [40] A. S. Oddy, *Three-Dimensional, Finite-Deformation, Thermal-Elasto-Plastic Finite Element Analysis*, Ph.D. Thesis, Carleton University, Ottawa, Canada, 1987.
- [41] J.F. Nye, *Physical Properties of Crystals*, Oxford University Press, London, 196-204, 1967.
- [42] D. Rahrer, *The Intensive Grading of a Three-Dimensional Finite Element Mesh to Determine the Stress Intensity Factor for a Crack*, B.Eng. Thesis, Carleton University, Ottawa, Canada, 1988.
- [43] J. L. Bentley, 'Multidimensional Binary Search Trees Used for Associative Searching', *Commun. ACM*, **18**, 9, 509-517 (1975).
- [44] J. A. Goldak, A. S. Oddy and J. M. McDill, Invited Paper, 'Recent Developments in Computational Weld Mechanics', *Euromech 221*, 1987, Lulæ, Sweeden, June 1987.
- [45] O. C. Zienkiewicz and A. Craig, 'Adaptive Refinement, Error Estimates, Multigrid Solution, and Hierarchic Finite Element Method Concepts', *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, John Wiley and Sons Ltd., Chichester, U.K., 3-23, 1986.
- [46] *PADL-2, Parts and Description Language*, Production Automation Laboratory, University of Rochester, Rochester, New York, 1983.
- [47] *INGRID, A Three-Dimensional Mesh Generator for Modeling Non-Linear Systems*, Lawrence Livermore National Laboratory, California, 1981.
- [48] J. A. Goldak, *PROTEUS, A Data Transfer Algorithm*, Carleton University, Ottawa, in progress.
- [49] A. Peano, 'Hierarchies of Conforming Finite Elements for Plane Elasticity and Plate Bending', *Comp. and Maths. with Appls.*, **2**, 211-224 (1976).

- [50] A. G. Peano, B. A. Szabo and A. K. Mehta, 'Self-Adaptive Finite Elements in Fracture Mechanics', *Comp. Meth. Appl. Mech. Eng.*, **16**, 69-80 (1978).
- [51] I. N. Katz, A. G. Peano and M. P. Rossow, 'Nodal Variables for Complete Conforming Finite Elements of Arbitrary Polynomial Order', *Comp. and Maths. with Appls.*, **4**, 85-112 (1978).
- [52] I. Babuška, I. N. Katz and B. A. Szabo, 'Hierarchic Families for the P-Version of the Finite Element Method', *Advances in Computer Methods for Partial Differential Equations, III*, IMACS North-Holland Pub. Co., 1979.
- [53] Z. Pammer, *PEXFE (P-Extension of the Finite Element Method)*, released by RE-ALCO, Budapest, Hungary.
- [54] W. C. Thacker, 'A Brief Review of Techniques for Generating Irregular Computational Grids', *Int. j. numer. methods. eng.*, **15**, 1335-1341 (1980).
- [55] J. C. Cavendish, 'Automatic Triangulation of Arbitrary Planar Domains for the Finite Element Method', *Int. j. numer. methods. eng.*, **8**, 679-696 (1974).
- [56] Z. Pammer, 'A Mesh Refinement Method for Transient Heat Conduction Problems Solved by the Finite Element Method', *Int. j. numer. methods. eng.*, **15**, 495-505 (1980).
- [57] R. E. Bank and A. H. Sherman, 'The Use of Adaptive Grid Refinement for Badly Behaved Elliptic Partial Differential Equations', *Mathematics and Computers in Simulation*, **XXII**, 18-24 (1980).
- [58] R. E. Bank, 'A Posteriori Error Estimates and Adaptive Local Mesh Refinement of Elliptic Partial Differential Equations', *Mathematical and Computational Methods in Seismic Exploration and Reservoir Modeling*, SIAM, Philadelphia, 240-241, 1986.
- [59] R. E. Bank, 'Analysis of a Local a posteriori Error Estimate for Elliptic Equations', *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, John Wiley and Sons Ltd., Chichester, U.K., 119-128, 1986.
- [60] R. E. Bank, A. H. Sherman and A. Weiser, 'Refinement Algorithms and Data Structures for Regular Mesh Refinement', *Scientific Computing*, IMACS North Holland Pub. Co., 1983.
- [61] M. C. Rivera, 'Adaptive Finite Element Refinement and Fully Irregular and Conform-

- ing Triangulations', *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, John Wiley and Sons Ltd., Chichester, U.K., 359-370, 1986.
- [62] M. C. Rivera, 'Design and Data Structure of Fully Adaptive, Multigrid, Finite-Element Software', *ACM Trans. Math. Software*, 10, 3, 242-264 (1984).
 - [63] M. C. Rivera, 'A Grid Generator Based on 4-Triangles Conforming Mesh-Refinement Algorithms', *Int. j. numer. methods. eng.*, 24, 1343-1354 (1987)
 - [64] R. Löhner, K. Morgan, O. C. Zienkiewicz, 'Adaptive Grid Refinement for the Compressible Euler Equations', *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, John Wiley and Sons Ltd., Chichester, U.K., 281-297, 1986.
 - [65] O. C. Zienkiewicz, R. Löhner, K. Morgan and J. Peraire, 'High-Speed Compressible Flow and Other Advection-Dominated Problems of Fluid Dynamics', *Finite Elements in Fluids*, 6, John Wiley and Sons Ltd., Chichester, U.K., 41-88, 1986
 - [66] I. Babuška and W. C. Rheinboldt, 'Error Estimates for Adaptive Finite Element Computations', *SIAM J. Numer. Anal.*, 15, 736-754 (1978).
 - [67] I. Babuška and W. C. Rheinboldt, 'Reliable Error Estimation and Mesh Adaptation for the Finite Element Method', *Computational Methods in Nonlinear Mechanics*, North-Holland Pub. Co., Amsterdam, 67-108, 1980.
 - [68] P. Zave and W. C. Rheinboldt, 'Design of an Adaptive, Parallel Finite-Element System', *ACM Trans. on Math. Software*, 5, 1, 1-17 (1979).
 - [69] L. Demkowicz, J. T. Oden, T. Strouboulis, 'An Adaptive P-Version Finite Element Method for Transient Flow Problems with Moving Boundaries', *Finite Elements in Fluids*, 6, John Wiley and Sons Ltd., Chichester, U.K., 291-306, 1986.
 - [70] K. Miller and R. N. Miller, 'Moving Finite Elements I', *SIAM J. Numer. Anal.*, 18, 6, 1019-1032 (1981).
 - [71] R. J. Gelinas, S. K. Doss and K. Miller, 'The Moving Finite Element Method: Applications to General Partial Differential Equations with Multiple Large Gradients', *J. Comp. Phys.*, 40, 202-248 (1981).
 - [72] K. Miller, 'Recent Results on Finite Element Methods with Moving Nodes', *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, John Wiley and Sons Ltd., Chichester, U.K., 325-338, 1986

- [73] Z-B. Kuang and S. N. Atluri, 'Temperature Field Due to a Moving Heat Sources: A Moving Mesh Finite Element Analysis', *Trans. ASME, J. Appl. Mech.*, **19** (1985).
- [74] A. R. Mitchell and B. M. Herbst, 'Adaptive Grids in Petrov-Galerkin Computations', *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, John Wiley and Sons Ltd., Chichester, U.K., 315-324, 1986.
- [75] M. J. Djomehri, 'The Moving Finite Element Method and Its Applications', *Mathematical and Computational Methods in Seismic Exploration and Reservoir Modeling*, SIAM, Philadelphia, 230-232, 1986.
- [76] T. Sussman and K-J. Bathe, 'The Gradient of the Finite Element Variational Indicator with Respect to Nodal Point Co-ordinates: An Explicit Calculation and Application in Fracture Mechanics and Mesh Optimization', *Int. j. numer. methods. eng.*, **21**, 763-774 (1985).
- [77] J. O. Hallquist, *DYNA3D Course Notes, Fall 1982, CE3151-12, UCID-19899-1, Rev. 1*, Lawrence Livermore National Laboratory, California, 1986.
- [78] T. Belytschko and T. J. R. Hughes, *Computational Methods for Transient Analysis*, **1**, North Holland Pub. Co., Amsterdam, 1983.
- [79] G. Strang and G. J. Fix, *An Analysis of the Finite Element Method*, Prentice-Hall, Englewood Cliffs, N.J., 155, 1973.
- [80] L. Meirovitch, *Computational Methods in Structural Dynamics*, Sijthoff and Noordhoff, Alphen aan den Rijn, The Netherlands, 365, 1980.
- [81] J. P. De; S. R. Gago, D. W. Kelly and O. C. Zienkiewicz, 'A Posteriori Error Analysis and Adaptive Processes in the Finite Element Method: Part II - Adaptive Mesh Refinement', *Int. j. numer. methods. eng.*, **19**, 1621-1656 (1983).
- [82] A. Brandt, 'Multi-Level Adaptive Solutions to Boundary Value Problems', *Math. Comput.*, **31**, 138, 333-390 (1977).
- [83] M. A. Yerry and M. S. Shepard, 'Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique', *Int. j. numer. methods eng.*, **20**, 1965-1990 (1984).
- [84] M. A. Yerry and M. S. Shepard, 'Automatic Mesh Generation for Three-Dimensional Solids', *Computers and Structures*, **20**, 31-39 (1985).
- [85] J. C. Cavendish, D. F. Field and W. H. Frey, 'An Approach to Automatic Three-

- Dimensional Finite Element Mesh Generation', *Int. j. numer. methods eng.*, **21**, 329-347, (1985).
- [86] M. S. Shepard, M. A. Yerry and P. L. Baehmann, 'Automatic Mesh Generation Allowing for Efficient A Priori and A Posteriori Mesh Refinement', *Comp. Meth. Appl. Mech. Eng.*, **55**, 161-180 (1986).
 - [87] P. L. Baehmann, S. L. Wittchen, M. S. Shepard, K.R. Grice, M. A. Yerry, 'Robust, Geometrically Based, Automatic, Two-Dimensional Mesh Generation', *Int. j. numer. methods eng.*, **24**, 1043-1078 (1987).
 - [88] N. Kikuchi, 'Adaptive Grid-Design Methods for Finite Element Analysis', *Comp. Meth. Appl. Mech. Eng.*, **55**, 129-160 (1986).
 - [89] R. E. Ewing, 'Adaptive Mesh Refinements in Large-Scale Fluid Flow Simulation', *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, John Wiley and Sons Ltd., Chichester, U.K., 299-314, 1986.
 - [90] M. Bièterman, J. E. Flaherty and P. K. Moore, 'Adaptive Refinement Methods for Non-linear Parabolic Partial Differential Equations', *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, John Wiley and Sons Ltd., Chichester, U.K., 339-358, 1986.
 - [91] E. J. Otoo, *Balanced Multidimensional Extendible Hashing Tree*, Extended Abstract, Carleton University, Ottawa, Canada, 1986.
 - [92] J. C. Díaz, R. E. Ewing, R. W. Jones, A.E. McDonald, D. U. von Rosenberg, and L. M. Uhler, 'Local Grid Refinement for Oil Recovery Simulation', *Mathematical and Computational Methods in Seismic Exploration and Reservoir Modeling*, SIAM, Philadelphia, 238-240, 1986.
 - [93] J. C. Díaz, R. E. Ewing, R. W. Jones, A.E. McDonald, D. U. von Rosenberg, L. M. Uhler, 'Self-Adaptive Local Grid Refinement for Time-Dependent Two-Dimensional Simulation', *Finite Elements in Fluids*, **6**, John Wiley and Sons Ltd., Chichester, U.K., 279-290, 1986.
 - [94] G. R. Johnson and R. A. Styrk, 'Dynamic Lagrangian Computations for Solids, with Variable Nodal Connectivity for Severe Distortions', *Int. j. numer. methods eng.*, **23**, 509-522 (1986).

- [95] A. A. G. Requicha and H. B. Voelcker, 'Solids Modelling: Current Status and Research Directions', *IEEE Computer Graphics and Applications*, 3, 7, 25-37, 1983.
- [96] N. Holtz, *An Implementation of the k-D tree*, Carleton University, Ottawa, 1986.
- [97] M. H. Overmars, 'Geometric Data Structures for Computer Graphics: an Overview', *Theoretical Foundations for Computer Graphics and CAD*, NATO Advanced Study Institute, El Ciocco, Italy, July 4-17, 1987.
- [98] D. P. Dobkin, 'Computational Geometry', *Theoretical Foundations for Computer Graphics and CAD*, NATO Advanced Study Institute, El Ciocco, Italy, July 4-17, 1987.
- [99] R. J. Wilson and L. W. Beinke, *Applications of Graph Theory*, Academic Press, London, 381-417, 1979.
- [100] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *Data Structures and Algorithms*, Addison-Wesley Pub. Co., Reading, Massachusetts, 1983.
- [101] K. Preiss, 'Checking the Topological Consistency of a Finite Element Mesh', *Int. j. numer. methods eng.*, 14, 1805-1812 (1979).
- [102] D. Downey, *Private Communication*, December 1987.
- [103] O. C. Zienkiewicz, *The Finite Element Method*, 3rd Ed., McGraw Hill Book Co., London, 162-164, 1979.
- [104] K-J. Bathe, *Finite Element Procedures in Engineering Analysis*, Prentice-Hall, Englewood Cliffs, N.J., 449-458, 1982.
- [105] J. K. Dukowicz, 'Conservative Rezoning (Remapping) for General Quadrilateral Meshes', *J. Comp. Phys.*, 54, 411-424 (1984).
- [106] *DYNA2D and DYNA3D*, Lawrence Livermore National Laboratory, California.
- [107] R. E. Ewing, 'Adaptive Local Grid Refinement', *Mathematical and Computational Methods in Seismic Exploration and Reservoir Modeling*, SIAM, Philadelphia, 235-236, 1986.
- [108] J. T. Oden, L. Demkowicz, T. Strouboulis and P. Devloo, 'Adaptive Methods for Problems in Solid and Fluid Mechanics', *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, John Wiley and Sons Ltd., Chichester, U.K., 249-280, 1986.

- [109] W. C. Rheinboldt, 'Error Estimates and Adaptive Techniques for Non-Linear Parameterized Equations', *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, John Wiley and Sons Ltd., Chichester, U.K., 163-180, 1986.
- [110] A. Knight, *A Smalltalk Prototype for Automatic Finite Element Mesh Refinement and Coarsening Using Graded Elements*, B.C.S. Thesis, Dept. Computer Science, Carleton University, Ottawa, April, 1987.
- [111] M. Bièterman, 'Control of Cost and Discretization Errors for Reaction-Diffusion Systems', *Mathematical and Computational Methods in Seismic Exploration and Reservoir Modeling*, SIAM, Philadelphia, 227-228, 1986.

Appendix I: Basis Functions and Local Derivatives 4- to 8-Node Quadrilateral

When node 8, $(\xi, \eta) = (1, 0)$, exists:

$$\begin{aligned} N_8 &= 1/2(1 + \xi)(1 - |\eta|) \\ \partial N_8 / \partial \xi &= 1/2(1 - |\eta|) \\ \text{If } \eta > 0 \text{ then } \partial N_8 / \partial \eta &= -1/2(1 + \xi) \\ \text{else } \partial N_8 / \partial \eta &= 1/2(1 + \xi) \end{aligned}$$

When node 7, $(\xi, \eta) = (0, -1)$, exists:

$$\begin{aligned} N_7 &= 1/2(1 - |\xi|)(1 - \eta) \\ \text{If } \xi > 0 \text{ then } \partial N_7 / \partial \xi &= -1/2(1 - \eta) \\ \text{else } \partial N_7 / \partial \xi &= 1/2(1 - \eta) \\ \partial N_7 / \partial \eta &= -1/2(1 - |\xi|) \end{aligned}$$

When node 6, $(\xi, \eta) = (-1, 0)$, exists:

$$\begin{aligned} N_6 &= 1/2(1 - \xi)(1 - |\eta|) \\ \partial N_6 / \partial \xi &= -1/2(1 - |\eta|) \\ \text{If } \eta > 0 \text{ then } \partial N_6 / \partial \eta &= -1/2(1 - \xi) \\ \text{else } \partial N_6 / \partial \eta &= 1/2(1 - \xi) \end{aligned}$$

When node 5, $(\xi, \eta) = (0, 1)$, exists:

$$\begin{aligned} N_5 &= 1/2(1 - |\xi|)(1 + \eta) \\ \text{If } \xi > 0 \text{ then } \partial N_5 / \partial \xi &= -1/2(1 + \eta) \\ \text{else } \partial N_5 / \partial \xi &= 1/2(1 + \eta) \\ \partial N_5 / \partial \eta &= 1/2(1 - |\xi|) \end{aligned}$$

Nodes 1 to 4 are mandatory, so there is no need to check if they exist:

Node 4, $(\xi, \eta) = (1, -1)$:

$$\begin{aligned} N_4 &= 1/4(1 + \xi)(1 - \eta) - 1/2(N_7 + N_8) \\ \partial N_4 / \partial \xi &= 1/4(1 - \eta) - 1/2(\partial N_7 / \partial \xi + \partial N_8 / \partial \xi) \\ \partial N_4 / \partial \eta &= -1/4(1 + \xi) - 1/2(\partial N_7 / \partial \eta + \partial N_8 / \partial \eta) \end{aligned}$$

Node 3, $(\xi, \eta) = (-1, -1)$:

$$\begin{aligned} N_3 &= 1/4(1 - \xi)(1 - \eta) - 1/2(N_6 + N_7) \\ \partial N_3 / \partial \xi &= -1/4(1 - \eta) - 1/2(\partial N_6 / \partial \xi + \partial N_7 / \partial \xi) \\ \partial N_3 / \partial \eta &= -1/4(1 - \xi) - 1/2(\partial N_6 / \partial \eta + \partial N_7 / \partial \eta) \end{aligned}$$

Node 2, $(\xi, \eta) = (-1, 1)$:

$$\begin{aligned} N_2 &= 1/4(1 - \xi)(1 + \eta) - 1/2(N_6 + N_8) \\ \partial N_2 / \partial \xi &= -1/4(1 + \eta) - 1/2(\partial N_6 / \partial \xi + \partial N_8 / \partial \xi) \\ \partial N_2 / \partial \eta &= 1/4(1 - \xi) - 1/2(\partial N_6 / \partial \eta + \partial N_8 / \partial \eta) \end{aligned}$$

Node 1, $(\xi, \eta) = (1, 1)$:

$$\begin{aligned} N_1 &= 1/4(1 + \xi)(1 + \eta) - 1/2(N_6 + N_8) \\ \partial N_1 / \partial \xi &= 1/4(1 + \eta) - 1/2(\partial N_6 / \partial \xi + \partial N_8 / \partial \xi) \\ \partial N_1 / \partial \eta &= 1/4(1 + \xi) - 1/2(\partial N_6 / \partial \eta + \partial N_8 / \partial \eta) \end{aligned}$$

Appendix II: 3-D Basis Functions and Local Derivatives 8- to 26-Node Hexahedron

When node 26, $(\xi, \eta, \zeta) = (0, 0, -1)$, exists :

$$N_{26} = 1/2(1 - |\xi|)(1 - |\eta|)(1 - \zeta)$$

If $\xi > 0$ then $\partial N_{26}/\partial \xi = -1/2(1 - |\eta|)(1 - \zeta)$

else $\partial N_{26}/\partial \xi = 1/2(1 - |\eta|)(1 - \zeta)$

If $\eta > 0$ then $\partial N_{26}/\partial \eta = -1/2(1 - |\xi|)(1 - \zeta)$

else $\partial N_{26}/\partial \eta = 1/2(1 - |\xi|)(1 - \zeta)$

$$\partial N_{26}/\partial \zeta = -1/2(1 - |\xi|)(1 - |\eta|)$$

When node 25, $(\xi, \eta, \zeta) = (0, 0, 1)$, exists :

$$N_{25} = 1/2(1 - |\xi|)(1 - |\eta|)(1 + \zeta)$$

If $\xi > 0$ then $\partial N_{25}/\partial \xi = -1/2(1 - |\eta|)(1 + \zeta)$

else $\partial N_{25}/\partial \xi = 1/2(1 - |\eta|)(1 + \zeta)$

If $\eta > 0$ then $\partial N_{25}/\partial \eta = -1/2(1 - |\xi|)(1 + \zeta)$

else $\partial N_{25}/\partial \eta = 1/2(1 - |\xi|)(1 + \zeta)$

$$\partial N_{25}/\partial \zeta = 1/2(1 - |\xi|)(1 - |\eta|)$$

When node 24, $(\xi, \eta, \zeta) = (0, -1, 0)$, exists :

$$N_{24} = 1/2(1 - |\xi|)(1 - \eta)(1 - |\zeta|)$$

If $\xi > 0$ then $\partial N_{24}/\partial \xi = -1/2(1 - \eta)(1 - |\zeta|)$

else $\partial N_{24}/\partial \xi = 1/2(1 - \eta)(1 - |\zeta|)$

$$\partial N_{24}/\partial \eta = -1/2(1 - |\xi|)(1 - |\zeta|)$$

If $\zeta > 0$ then $\partial N_{24}/\partial \zeta = -1/2(1 - |\xi|)(1 - \eta)$

else $\partial N_{24}/\partial \zeta = 1/2(1 - |\xi|)(1 - \eta)$

When node 23, $(\xi, \eta, \zeta) = (0, 1, 0)$, exists :

$$N_{23} = 1/2(1 - |\xi|)(1 + \eta)(1 - |\zeta|)$$

If $\xi > 0$ then $\partial N_{23}/\partial \xi = -1/2(1 + \eta)(1 - |\zeta|)$

else $\partial N_{23}/\partial \xi = 1/2(1 + \eta)(1 - |\zeta|)$

$$\partial N_{23}'/\partial \eta = 1/2(1 - |\xi|)(1 - |\zeta|)$$

If $\zeta > 0$ then $\partial N_{23}/\partial \zeta = -1/2(1 - |\xi|)(1 + \eta)$

else $\partial N_{23}/\partial \zeta = 1/2(1 - |\xi|)(1 + \eta)$

When node 22, $(\xi, \eta, \zeta) = (-1, 0, 0)$, exists :

$$N_{22} = 1/2(1 - \xi)(1 - |\eta|)(1 - |\zeta|)$$

$$\partial N_{22}/\partial \xi = -1/2(1 - |\eta|)(1 - |\zeta|)$$

If $\eta > 0$ then $\partial N_{22}/\partial \eta = -1/2(1 - \xi)(1 - |\zeta|)$

else $\partial N_{22}/\partial \eta = 1/2(1 - \xi)(1 - |\zeta|)$

If $\zeta > 0$ then $\partial N_{22}/\partial \zeta = -1/2(1 - \xi)(1 - |\eta|)$

else $\partial N_{22}/\partial \zeta = 1/2(1 - \xi)(1 - |\eta|)$

When node 21, $(\xi, \eta, \zeta) = (1, 0, 0)$, exists :

$$N_{21} = 1/2(1 + \xi)(1 - |\eta|)(1 - |\zeta|)$$

$$\partial N_{21}/\partial \xi = 1/2(1 - |\eta|)(1 - |\zeta|)$$

if $\eta > 0$ then $\partial N_{21}/\partial \eta = -1/2(1 + \xi)(1 - |\zeta|)$

else $\partial N_{21}/\partial \eta = 1/2(1 + \xi)(1 - |\zeta|)$

If $\zeta > 0$ then $\partial N_{21}/\partial \zeta = -1/2(1 + \xi)(1 - |\eta|)$

else $\partial N_{21}/\partial \zeta = 1/2(1 + \xi)(1 - |\eta|)$

When node 20, $(\xi, \eta, \zeta) = (1, -1, 0)$, exists :

$$N_{20} = 1/4(1 + \xi)(1 - \eta)(1 - |\zeta|) - 1/2(N_{21} + N_{24})$$

$$\partial N_{20}/\partial \xi = 1/4(1 - \eta)(1 - |\zeta|) - 1/2(\partial N_{21}/\partial \xi + \partial N_{24}/\partial \xi)$$

$$\partial N_{20}/\partial \eta = -1/4(1 + \xi)(1 - |\zeta|) - 1/2(\partial N_{21}/\partial \eta + \partial N_{24}/\partial \eta)$$

If $\zeta > 0$ then $\partial N_{20}/\partial \zeta = -1/4(1 + \xi)(1 - \eta) - 1/2(\partial N_{21}/\partial \zeta + \partial N_{24}/\partial \zeta)$

else $\partial N_{20}/\partial \zeta = 1/4(1 + \xi)(1 - \eta) - 1/2(\partial N_{21}/\partial \zeta + \partial N_{24}/\partial \zeta)$

When node 19, $(\xi, \eta, \zeta) = (-1, -1, 0)$, exists :

$$\begin{aligned}
 N_{19} &= 1/4(1 - \xi)(1 - \eta)(1 - |\zeta|) - 1/2(N_{22} + N_{24}) \\
 \partial N_{19}/\partial \xi &= -1/4(1 - \eta)(1 - |\zeta|) - 1/2(\partial N_{22}/\partial \xi + \partial N_{24}/\partial \xi) \\
 \partial N_{19}/\partial \eta &= -1/4(1 - \xi)(1 - |\zeta|) - 1/2(\partial N_{22}/\partial \eta + \partial N_{24}/\partial \eta) \\
 \text{If } \zeta > 0 \text{ then } \partial N_{19}/\partial \zeta &= -1/4(1 - \xi)(1 - \eta) - 1/2(\partial N_{22}/\partial \zeta + \partial N_{24}/\partial \zeta) \\
 \text{else } \partial N_{19}/\partial \zeta &= 1/4(1 - \xi)(1 - \eta) - 1/2(\partial N_{22}/\partial \zeta + \partial N_{24}/\partial \zeta)
 \end{aligned}$$

When node 18, $(\xi, \eta, \zeta) = (-1, 1, 0)$, exists :

$$\begin{aligned}
 N_{18} &= 1/4(1 - \xi)(1 + \eta)(1 - |\zeta|) - 1/2(N_{22} + N_{23}) \\
 \partial N_{18}/\partial \xi &= -1/4(1 + \eta)(1 - |\zeta|) - 1/2(\partial N_{22}/\partial \xi + \partial N_{23}/\partial \xi) \\
 \partial N_{18}/\partial \eta &= 1/4(1 - \xi)(1 - |\zeta|) - 1/2(\partial N_{22}/\partial \eta + \partial N_{23}/\partial \eta) \\
 \text{If } \zeta > 0 \text{ then } \partial N_{18}/\partial \zeta &= -1/4(1 - \xi)(1 + \eta) - 1/2(\partial N_{22}/\partial \zeta + \partial N_{23}/\partial \zeta) \\
 \text{else } \partial N_{18}/\partial \zeta &= 1/4(1 - \xi)(1 + \eta) - 1/2(\partial N_{22}/\partial \zeta + \partial N_{23}/\partial \zeta)
 \end{aligned}$$

When node 17, $(\xi, \eta, \zeta) = (1, 1, 0)$, exists :

$$\begin{aligned}
 N_{17} &= 1/4(1 + \xi)(1 + \eta)(1 - |\zeta|) - 1/2(N_{21} + N_{23}) \\
 \partial N_{17}/\partial \xi &= 1/4(1 + \eta)(1 - |\zeta|) - 1/2(\partial N_{21}/\partial \xi + \partial N_{23}/\partial \xi) \\
 \partial N_{17}/\partial \eta &= 1/4(1 + \xi)(1 - |\zeta|) - 1/2(\partial N_{21}/\partial \eta + \partial N_{23}/\partial \eta) \\
 \text{If } \zeta > 0 \text{ then } \partial N_{17}/\partial \zeta &= -1/4(1 + \xi)(1 + \eta) - 1/2(\partial N_{21}/\partial \zeta + \partial N_{23}/\partial \zeta) \\
 \text{else } \partial N_{17}/\partial \zeta &= 1/4(1 + \xi)(1 + \eta) - 1/2(\partial N_{21}/\partial \zeta + \partial N_{23}/\partial \zeta)
 \end{aligned}$$

When node 16, $(\xi, \eta, \zeta) = (1, 0, -1)$, exists :

$$\begin{aligned}
 N_{16} &= 1/4(1 + \xi)(1 - |\eta|)(1 - \zeta) - 1/2(N_{21} + N_{26}) \\
 \partial N_{16}/\partial \xi &= 1/4(1 - |\eta|)(1 - \zeta) - 1/2(\partial N_{21}/\partial \xi + \partial N_{26}/\partial \xi) \\
 \text{If } \eta > 0 \text{ then } \partial N_{16}/\partial \eta &= -1/4(1 + \xi)(1 - \zeta) - 1/2(\partial N_{21}/\partial \eta + \partial N_{26}/\partial \eta) \\
 \text{else } \partial N_{16}/\partial \eta &= 1/4(1 + \xi)(1 - \zeta) - 1/2(\partial N_{21}/\partial \eta + \partial N_{26}/\partial \eta) \\
 \partial N_{16}/\partial \zeta &= -1/4(1 + \xi)(1 - |\eta|) - 1/2(\partial N_{21}/\partial \zeta + \partial N_{26}/\partial \zeta)
 \end{aligned}$$

When node 15, $(\xi, \eta, \varsigma) = (0, -1, -1)$, exists :

$$\begin{aligned}
 N_{15} &= 1/4(1 - |\xi|)(1 - \eta)(1 - \varsigma) - 1/2(N_{24} + N_{26}) \\
 \text{If } \xi > 0 \text{ then } \quad \partial N_{15}/\partial \xi &= -1/4(1 - \eta)(1 - \varsigma) - 1/2(\partial N_{24}/\partial \xi + \partial N_{26}/\partial \xi) \\
 \text{else} \quad \partial N_{15}/\partial \xi &= 1/4(1 - \eta)(1 - \varsigma) - 1/2(\partial N_{24}/\partial \xi + \partial N_{26}/\partial \xi) \\
 \partial N_{15}/\partial \eta &= -1/4(1 - |\xi|)(1 - \varsigma) - 1/2(\partial N_{24}/\partial \eta + \partial N_{26}/\partial \eta) \\
 \partial N_{15}/\partial \varsigma &= -1/4(1 - |\xi|)(1 - \eta) - 1/2(\partial N_{24}/\partial \varsigma + \partial N_{26}/\partial \varsigma)
 \end{aligned}$$

When node 14, $(\xi, \eta, \varsigma) = (-1, 0, -1)$, exists :

$$\begin{aligned}
 N_{14} &= 1/4(1 - \xi)(1 - |\eta|)(1 - \varsigma) - 1/2(N_{22} + N_{26}) \\
 \partial N_{14}/\partial \xi &= -1/4(1 - |\eta|)(1 - \varsigma) - 1/2(\partial N_{22}/\partial \xi + \partial N_{26}/\partial \xi) \\
 \text{If } \eta > 0 \text{ then } \quad \partial N_{14}/\partial \eta &= -1/4(1 - \xi)(1 - \varsigma) - 1/2(\partial N_{22}/\partial \eta + \partial N_{26}/\partial \eta) \\
 \text{else} \quad \partial N_{14}/\partial \eta &= 1/4(1 - \xi)(1 - \varsigma) - 1/2(\partial N_{22}/\partial \eta + \partial N_{26}/\partial \eta) \\
 \partial N_{14}/\partial \varsigma &= -1/4(1 - \xi)(1 - |\eta|) - 1/2(\partial N_{22}/\partial \varsigma + \partial N_{26}/\partial \varsigma)
 \end{aligned}$$

When node 13, $(\xi, \eta, \varsigma) = (0, 1, -1)$, exists :

$$\begin{aligned}
 N_{13} &= 1/4(1 - |\xi|)(1 + \eta)(1 - \varsigma) - 1/2(N_{23} + N_{26}) \\
 \text{If } \xi > 0 \text{ then } \quad \partial N_{13}/\partial \xi &= -1/4(1 + \eta)(1 - \varsigma) - 1/2(\partial N_{23}/\partial \xi + \partial N_{26}/\partial \xi) \\
 \text{else} \quad \partial N_{13}/\partial \xi &= 1/4(1 + \eta)(1 - \varsigma) - 1/2(\partial N_{23}/\partial \xi + \partial N_{26}/\partial \xi) \\
 \partial N_{13}/\partial \eta &= 1/4(1 - |\xi|)(1 - \varsigma) - 1/2(\partial N_{23}/\partial \eta + \partial N_{26}/\partial \eta) \\
 \partial N_{13}/\partial \varsigma &= -1/4(1 - |\xi|)(1 + \eta) - 1/2(\partial N_{23}/\partial \varsigma + \partial N_{26}/\partial \varsigma)
 \end{aligned}$$

When node 12, $(\xi, \eta, \varsigma) = (1, 0, 1)$, exists :

$$\begin{aligned}
 N_{12} &= 1/4(1 + \xi)(1 - |\eta|)(1 + \varsigma) - 1/2(N_{21} + N_{25}) \\
 \partial N_{12}/\partial \xi &= 1/4(1 - |\eta|)(1 + \varsigma) - 1/2(\partial N_{21}/\partial \xi + \partial N_{25}/\partial \xi) \\
 \text{If } \eta > 0 \text{ then } \quad \partial N_{12}/\partial \eta &= -1/4(1 + \xi)(1 + \varsigma) - 1/2(\partial N_{21}/\partial \eta + \partial N_{25}/\partial \eta) \\
 \text{else} \quad \partial N_{12}/\partial \eta &= 1/4(1 + \xi)(1 + \varsigma) - 1/2(\partial N_{21}/\partial \eta + \partial N_{25}/\partial \eta) \\
 \partial N_{12}/\partial \varsigma &= 1/4(1 + \xi)(1 - |\eta|) - 1/2(\partial N_{21}/\partial \varsigma + \partial N_{25}/\partial \varsigma)
 \end{aligned}$$

When node 11, $(\xi, \eta, \zeta) = (0, -1, 1)$, exists :

$$\begin{aligned}
 N_{11} &= 1/4(1 - |\xi|)(1 - \eta)(1 + \zeta) - 1/2(N_{24} + N_{25}) \\
 \text{If } \xi > 0 \text{ then } \quad \partial N_{11}/\partial \xi &= -1/4(1 - \eta)(1 + \zeta) - 1/2(\partial N_{24}/\partial \xi + \partial N_{25}/\partial \xi) \\
 \text{else} \quad \partial N_{11}/\partial \xi &= 1/4(1 - \eta)(1 + \zeta) - 1/2(\partial N_{24}/\partial \xi + \partial N_{25}/\partial \xi) \\
 \partial N_{11}/\partial \eta &= -1/4(1 - |\xi|)(1 + \zeta) - 1/2(\partial N_{24}/\partial \eta + \partial N_{25}/\partial \eta) \\
 \partial N_{11}/\partial \zeta &= 1/4(1 - |\xi|)(1 - \eta) - 1/2(\partial N_{24}/\partial \zeta + \partial N_{25}/\partial \zeta)
 \end{aligned}$$

When node 10, $(\xi, \eta, \zeta) = (-1, 0, 1)$, exists :

$$\begin{aligned}
 N_{10} &= 1/4(1 - \xi)(1 - |\eta|)(1 + \zeta) - 1/2(N_{22} + N_{25}) \\
 \partial N_{10}/\partial \xi &= -1/4(1 - |\eta|)(1 + \zeta) - 1/2(\partial N_{22}/\partial \xi + \partial N_{25}/\partial \xi) \\
 \text{If } \eta > 0 \text{ then } \quad \partial N_{10}/\partial \eta &= -1/4(1 - \xi)(1 + \zeta) - 1/2(\partial N_{22}/\partial \eta + \partial N_{25}/\partial \eta) \\
 \text{else} \quad \partial N_{10}/\partial \eta &= 1/4(1 - \xi)(1 + \zeta) - 1/2(\partial N_{22}/\partial \eta + \partial N_{25}/\partial \eta) \\
 \partial N_{10}/\partial \zeta &= 1/4(1 - \xi)(1 - |\eta|) - 1/2(\partial N_{22}/\partial \zeta + \partial N_{25}/\partial \zeta)
 \end{aligned}$$

When node 9, $(\xi, \eta, \zeta) = (0, 1, 1)$, exists :

$$\begin{aligned}
 N_9 &= 1/4(1 - |\xi|)(1 + \eta)(1 + \zeta) - 1/2(N_{23} + N_{25}) \\
 \text{If } \xi > 0 \text{ then } \quad \partial N_9/\partial \xi &= -1/4(1 + \eta)(1 + \zeta) - 1/2(\partial N_{23}/\partial \xi + \partial N_{25}/\partial \xi) \\
 \text{else} \quad \partial N_9/\partial \xi &= 1/4(1 + \eta)(1 + \zeta) - 1/2(\partial N_{23}/\partial \xi + \partial N_{25}/\partial \xi) \\
 \partial N_9/\partial \eta &= 1/4(1 - |\xi|)(1 + \zeta) - 1/2(\partial N_{23}/\partial \eta + \partial N_{25}/\partial \eta) \\
 \partial N_9/\partial \zeta &= 1/4(1 - |\xi|)(1 + \eta) - 1/2(\partial N_{23}/\partial \zeta + \partial N_{25}/\partial \zeta)
 \end{aligned}$$

Nodes 1 to 8 are mandatory, so there is no need to check to see if they exist.

Node 8, $(\xi, \eta, \zeta) = (1, -1, -1)$:

$$\begin{aligned}
 N_8 &= 1/8(1 + \xi)(1 - \eta)(1 - \zeta) - 1/2(N_{15} + N_{16} + N_{20}) - 1/4(N_{21} + N_{24} + N_{26}) \\
 \partial N_8/\partial \xi &= 1/8(1 - \eta)(1 - \zeta) - 1/2(\partial N_{15}/\partial \xi + \partial N_{16}/\partial \xi + \partial N_{20}/\partial \xi) \\
 &\quad - 1/4(\partial N_{21}/\partial \xi + \partial N_{24}/\partial \xi + \partial N_{26}/\partial \xi) \\
 \partial N_8/\partial \eta &= -1/8(1 + \xi)(1 - \zeta) - 1/2(\partial N_{15}/\partial \eta + \partial N_{16}/\partial \eta + \partial N_{20}/\partial \eta) \\
 &\quad - 1/4(\partial N_{21}/\partial \eta + \partial N_{24}/\partial \eta + \partial N_{26}/\partial \eta) \\
 \partial N_8/\partial \zeta &= -1/8(1 + \xi)(1 - \eta) - 1/2(\partial N_{15}/\partial \zeta + \partial N_{16}/\partial \zeta + \partial N_{20}/\partial \zeta) \\
 &\quad - 1/4(\partial N_{21}/\partial \zeta + \partial N_{24}/\partial \zeta + \partial N_{26}/\partial \zeta)
 \end{aligned}$$

Node 7, $(\xi, \eta, \zeta) = (-1, -1, -1)$:

$$N_7 = 1/8(1 - \xi)(1 - \eta)(1 - \zeta) - 1/2(N_{14} + N_{15} + N_{19}) - 1/4(N_{22} + N_{24} + N_{26})$$

$$\begin{aligned} \partial N_7 / \partial \xi = & -1/8(1 - \eta)(1 - \zeta) - 1/2(\partial N_{14} / \partial \xi + \partial N_{15} / \partial \xi + \partial N_{19} / \partial \xi) \\ & - 1/4(\partial N_{22} / \partial \xi + \partial N_{24} / \partial \xi + \partial N_{26} / \partial \xi) \end{aligned}$$

$$\begin{aligned} \partial N_7 / \partial \eta = & -1/8(1 - \xi)(1 - \zeta) - 1/2(\partial N_{14} / \partial \eta + \partial N_{15} / \partial \eta + \partial N_{19} / \partial \eta) \\ & - 1/4(\partial N_{22} / \partial \eta + \partial N_{24} / \partial \eta + \partial N_{26} / \partial \eta) \end{aligned}$$

$$\begin{aligned} \partial N_7 / \partial \zeta = & -1/8(1 - \xi)(1 - \eta) - 1/2(\partial N_{14} / \partial \zeta + \partial N_{15} / \partial \zeta + \partial N_{19} / \partial \zeta) \\ & - 1/4(\partial N_{22} / \partial \zeta + \partial N_{24} / \partial \zeta + \partial N_{26} / \partial \zeta) \end{aligned}$$

Node 6, $(\xi, \eta, \zeta) = (-1, 1, -1)$:

$$N_6 = 1/8(1 - \xi)(1 + \eta)(1 - \zeta) - 1/2(N_{13} + N_{14} + N_{18}) - 1/4(N_{22} + N_{23} + N_{26})$$

$$\begin{aligned} \partial N_6 / \partial \xi = & -1/8(1 + \eta)(1 - \zeta) - 1/2(\partial N_{13} / \partial \xi + \partial N_{14} / \partial \xi + \partial N_{18} / \partial \xi) \\ & - 1/4(\partial N_{22} / \partial \xi + \partial N_{23} / \partial \xi + \partial N_{26} / \partial \xi) \end{aligned}$$

$$\begin{aligned} \partial N_6 / \partial \eta = & 1/8(1 - \xi)(1 - \zeta) - 1/2(\partial N_{13} / \partial \eta + \partial N_{14} / \partial \eta + \partial N_{18} / \partial \eta) \\ & - 1/4(\partial N_{22} / \partial \eta + \partial N_{23} / \partial \eta + \partial N_{26} / \partial \eta) \end{aligned}$$

$$\begin{aligned} \partial N_6 / \partial \zeta = & -1/8(1 - \xi)(1 + \eta) - 1/2(\partial N_{13} / \partial \zeta + \partial N_{14} / \partial \zeta + \partial N_{18} / \partial \zeta) \\ & - 1/4(\partial N_{22} / \partial \zeta + \partial N_{23} / \partial \zeta + \partial N_{26} / \partial \zeta) \end{aligned}$$

Node 5, $(\xi, \eta, \zeta) = (1, 1, -1)$:

$$N_5 = 1/8(1 + \xi)(1 + \eta)(1 - \zeta) - 1/2(N_{13} + N_{16} + N_{17}) - 1/4(N_{21} + N_{23} + N_{26})$$

$$\begin{aligned} \partial N_5 / \partial \xi = & 1/8(1 + \eta)(1 - \zeta) - 1/2(\partial N_{13} / \partial \xi + \partial N_{16} / \partial \xi + \partial N_{17} / \partial \xi) \\ & - 1/4(\partial N_{21} / \partial \xi + \partial N_{23} / \partial \xi + \partial N_{26} / \partial \xi) \end{aligned}$$

$$\begin{aligned} \partial N_5 / \partial \eta = & 1/8(1 + \xi)(1 - \zeta) - 1/2(\partial N_{13} / \partial \eta + \partial N_{16} / \partial \eta + \partial N_{17} / \partial \eta) \\ & - 1/4(\partial N_{21} / \partial \eta + \partial N_{23} / \partial \eta + \partial N_{26} / \partial \eta) \end{aligned}$$

$$\begin{aligned} \partial N_5 / \partial \zeta = & -1/8(1 + \xi)(1 + \eta) - 1/2(\partial N_{13} / \partial \zeta + \partial N_{16} / \partial \zeta + \partial N_{17} / \partial \zeta) \\ & - 1/4(\partial N_{21} / \partial \zeta + \partial N_{23} / \partial \zeta + \partial N_{26} / \partial \zeta) \end{aligned}$$

Node 4, $(\xi, \eta, \zeta) = (1, -1, 1)$:

$$N_4 = 1/8(1 + \xi)(1 - \eta)(1 + \zeta) - 1/2(N_{11} + N_{12} + N_{20}) - 1/4(N_{21} + N_{24} + N_{25})$$

$$\begin{aligned} \partial N_4 / \partial \xi = & 1/8(1 - \eta)(1 + \zeta) - 1/2(\partial N_{11} / \partial \xi + \partial N_{12} / \partial \xi + \partial N_{20} / \partial \xi) \\ & - 1/4(\partial N_{21} / \partial \xi + \partial N_{24} / \partial \xi + \partial N_{25} / \partial \xi) \end{aligned}$$

$$\begin{aligned} \partial N_4 / \partial \eta = & -1/8(1 + \xi)(1 + \zeta) - 1/2(\partial N_{11} / \partial \eta + \partial N_{12} / \partial \eta + \partial N_{20} / \partial \eta) \\ & - 1/4(\partial N_{21} / \partial \eta + \partial N_{24} / \partial \eta + \partial N_{25} / \partial \eta) \end{aligned}$$

$$\begin{aligned} \partial N_4 / \partial \zeta = & 1/8(1 + \xi)(1 - \eta) - 1/2(\partial N_{11} / \partial \zeta + \partial N_{12} / \partial \zeta + \partial N_{20} / \partial \zeta) \\ & - 1/4(\partial N_{21} / \partial \zeta + \partial N_{24} / \partial \zeta + \partial N_{25} / \partial \zeta) \end{aligned}$$

Node 3, $(\xi, \eta, \zeta) = (-1, -1, 1)$:

$$N_3 = 1/8(1 - \xi)(1 - \eta)(1 + \zeta) - 1/2(N_{10} + N_{11} + N_{19}) - 1/4(N_{22} + N_{24} + N_{25})$$

$$\begin{aligned} \partial N_3 / \partial \xi = & -1/8(1 - \eta)(1 + \zeta) - 1/2(\partial N_{10} / \partial \xi + \partial N_{11} / \partial \xi + \partial N_{19} / \partial \xi) \\ & - 1/4(\partial N_{22} / \partial \xi + \partial N_{24} / \partial \xi + \partial N_{25} / \partial \xi) \end{aligned}$$

$$\begin{aligned} \partial N_3 / \partial \eta = & -1/8(1 - \xi)(1 + \zeta) - 1/2(\partial N_{10} / \partial \eta + \partial N_{11} / \partial \eta + \partial N_{19} / \partial \eta) \\ & - 1/4(\partial N_{22} / \partial \eta + \partial N_{24} / \partial \eta + \partial N_{25} / \partial \eta) \end{aligned}$$

$$\begin{aligned} \partial N_3 / \partial \zeta = & 1/8(1 - \xi)(1 - \eta) - 1/2(\partial N_{10} / \partial \zeta + \partial N_{11} / \partial \zeta + \partial N_{19} / \partial \zeta) \\ & - 1/4(\partial N_{22} / \partial \zeta + \partial N_{24} / \partial \zeta + \partial N_{25} / \partial \zeta) \end{aligned}$$

Node 2, $(\xi, \eta, \zeta) = (-1, 1, 1)$:

$$N_2 = 1/8(1 - \xi)(1 + \eta)(1 + \zeta) - 1/2(N_9 + N_{10} + N_{18}) - 1/4(N_{22} + N_{23} + N_{25})$$

$$\begin{aligned} \partial N_2 / \partial \xi = & -1/8(1 + \eta)(1 + \zeta) - 1/2(\partial N_9 / \partial \xi + \partial N_{10} / \partial \xi + \partial N_{18} / \partial \xi) \\ & - 1/4(\partial N_{22} / \partial \xi + \partial N_{23} / \partial \xi + \partial N_{25} / \partial \xi) \end{aligned}$$

$$\begin{aligned} \partial N_2 / \partial \eta = & 1/8(1 - \xi)(1 + \zeta) - 1/2(\partial N_9 / \partial \eta + \partial N_{10} / \partial \eta + \partial N_{18} / \partial \eta) \\ & - 1/4(\partial N_{22} / \partial \eta + \partial N_{23} / \partial \eta + \partial N_{25} / \partial \eta) \end{aligned}$$

$$\begin{aligned} \partial N_2 / \partial \zeta = & 1/8(1 - \xi)(1 + \eta) - 1/2(\partial N_9 / \partial \zeta + \partial N_{10} / \partial \zeta + \partial N_{18} / \partial \zeta) \\ & - 1/4(\partial N_{22} / \partial \zeta + \partial N_{23} / \partial \zeta + \partial N_{25} / \partial \zeta) \end{aligned}$$

Node 1, $(\xi, \eta, \zeta) = (1, 1, 1)$:

$$N_1 = 1/8(1 + \xi)(1 + \eta)(1 + \zeta) - 1/2(N_9 + N_{12} + N_{17}) - 1/4(N_{21} + N_{23} + N_{25})$$

$$\begin{aligned} \partial N_1 / \partial \xi = & 1/8(1 + \eta)(1 + \zeta) - 1/2(\partial N_9 / \partial \xi + \partial N_{12} / \partial \xi + \partial N_{17} / \partial \xi) \\ & - 1/4(\partial N_{21} / \partial \xi + \partial N_{23} / \partial \xi + \partial N_{25} / \partial \xi) \end{aligned}$$

$$\begin{aligned} \partial N_1 / \partial \eta = & 1/8(1 + \xi)(1 + \zeta) - 1/2(\partial N_9 / \partial \eta + \partial N_{12} / \partial \eta + \partial N_{17} / \partial \eta) \\ & - 1/4(\partial N_{21} / \partial \eta + \partial N_{23} / \partial \eta + \partial N_{25} / \partial \eta) \end{aligned}$$

$$\begin{aligned} \partial N_1 / \partial \zeta = & 1/8(1 + \xi)(1 + \eta) - 1/2(\partial N_9 / \partial \zeta + \partial N_{12} / \partial \zeta + \partial N_{17} / \partial \zeta) \\ & - 1/4(\partial N_{21} / \partial \zeta + \partial N_{23} / \partial \zeta + \partial N_{25} / \partial \zeta) \end{aligned}$$

Appendix III: Graphics

One of the graphics postprocessing programs for ASGARD¹ is TAURUS.² It only recognizes 4-node linear quadrilaterals and 8-node linear bricks. Therefore, special code development was required for the grading elements. The first step was to do the plotting for only the corner nodes and their variables. This can be useful but tends to produce discontinuous contours of temperature and displacement and exaggerates the discontinuous nature of the strain and stress fields. Therefore, it was decided that a second capability would be added to convert the subelements to regular 4-node quadrilaterals and 8-node bricks for use with TAURUS.

The technique for creating subelements is straight-forward. The grading elements are examined at regular call to the routines which create the ASGARD files for graphics. Each element is then divided into new elements, based on the number of subelements that it has. For example, a 5-node quadrilateral is broken into two, 4-node quadrilaterals. The topology of the new elements is controlled by data blocks based on the local numbering of the element. Obviously, new nodes must sometimes be created. The values of the variables at these nodes must be found. Interpolation using the basis functions establishes the co-ordinates, temperatures and displacements of the new nodes.

A critical part of the procedure is the use of a spatially addressable k-D tree. When a new node is created for one element, it is stored in the k-D tree. The global co-ordinates of the node provide the key to the tree. At each subsequent interpolation for a new node, the tree is checked to see if the node has already been created. If it has, the node number is retrieved from the tree for use with the co-ordinates and topology.

A partial map of the more significant calls is presented below. A brief list of the pertinent subroutines and their descriptions are also included in this Appendix so subsequent implementations of the elements can also make use the graphics capabilities. It should

¹ ASGARD is a general purpose, nonlinear, transient finite element package written at Carleton University.

² TAURUS is a postprocessor written by D. Stillman at Lawrence Livermore National Laboratory.

be noted that the new elements and their topologies etc., cannot be used in further finite element analyses because they are not compatible in a finite element sense.

ELFILE	- NUGNU	- NUGNUGR	- DOMAIN2D	- PTNEW	
				- CORTBLE2D	- ADVAR2D
					- POINTS2D
					- PTGET
					- PTPUT
			- DOMAIN3D	- PTNEW	
				- CORTBLE3D	- ADVAR3D
					- POINTS3D
					- PTGET
					- PTPUT
ELFILE or	- NUGNUQ	- NUGNUQGR	- TMPTBL2D	- ADVAR2D	
FRONT or				- NWTMP2D	
CONOUT					
			- TMPTBL3D	- ADVAR3D	
				- NWTMP3D	
			- DISTBL2D	- ADVAR2D	
				- NWDIS2D	
			- DISTBL3D	- ADVAR3D	
				- NWDIS3D	

ADVAR2D Called by CORTBL2D, DISTBL2D, TMPTBL2D

ADVAR3D Called by CORTBL3D, DISTBL3D, TMPTBL3D

ADVAR2D and ADVAR3D calculate the shape functions for the 2-D and 3-D elements when a variable is being added.

CORTBL2D Called by DOMAIN2D

 Calls ADVAR2D and POINTS2D

 Calls PTGET and PTPUT (uses k-D tree
 to check for nodes)

CORTBL3D Called by DOMAIN3D

 Calls ADVAR3D and POINTS3D

 Calls PTGET and PTPUT (uses k-D tree
 to check for nodes)

CORTBL2D and CORTBL3D evaluate the shape functions (by calling ADVAR2D or ADVAR3D) at the new co-ordinates and find the global co-ordinates for the new node. Using the global co-ordinates as a key, a k-D tree (CORTBL) is checked to see if the node exists. If it already exists, the correct node number for the point is passed back. If it does not exist, it is stored in the tree.

DISTBL2D Called by NUGNUQGR

 Calls NWDIS2D

DISTBL3D Called by NUGNUQGR

 Calls NWDIS3D

DISTBL2D and DISTBL3D evaluate displacements at new nodes which are created when grading elements are divided into subelements.

DOMAIN2D Called by NUGNUGR

 Calls PTNEW - to open the k-D tree (CORTBL)

 Calls CORTBL2D

DOMAIN3D Called by NUGNUGR

 Calls PTNEW - to open the k-D tree (CORTBL)

 Calls CORTBL3D

DOMAIN2D and DOMAIN3D use information from a WRITE in SUBDOM3D (UNIT16) to identify and, if necessary, create the nodes and elements so TAURUS will plot temperatures and contours for the grading elements. Each 4- to 8-node element is divided into some number of new 4-node quadrilaterals, based on the number of subelements in the element. Each 8- to 26-node element is divided into some number of new 8-node bricks, based on the number of subelements in the element. This information is placed in a list of new topology.

The routine also uses PTNEW to open a k-D tree in which new nodal co-ordinates are stored. The k-D

tree is filled in either CORTBL2D or CORTBL3D. Once the k-D tree has been filled, the information is extracted and put in an array NCOORD using PTGET. NCOORD in turn is passed back to NUGNUGR, and then to NUGNU for plotting the subelements.

NUGNU	Called by ELFILE
	Calls NUGNUGR

NUGNU handles the geometry for regular elements and for grading elements based on their vertex nodes. NUGNUGR is then called to do graphics for the subelements.

NUGNUGR	Called by NUGNU
	Calls DOMAIN2D
	Calls DOMAIN3D

NUGNUGR was created to extract the subelements from the grading quadrilaterals (types 145 and 147) and from the grading bricks (types 146, 148 and 149). It is a copy of NUGNU with appropriate changes. NUGNU produces a compatible data file called GNU_DATA describing the ASGARD model. It can be used with the old ASGARD postprocessor GNU and with TAURUS. NUGNUGR writes to a compatible file GNUGR, which can be used with either GNU or TAURUS.

NUGNUQ	Called by FRONT (thermal analysis)
	Called by CONOUT (stress analysis)
	Called by ELFILE (in a thermal stress, records temperatures)
	Calls NUGNUQGR for grading bricks
NUGNUQGR	Called by NUGNUQ
	Calls TMPTBL2D for 2-D thermal analysis and thermal stress
	Calls DISTBL2D for 2-D stress and thermal stress
	Calls TMPTBL3D for 3-D thermal analysis and thermal stress
	Calls DISTBL3D for 3-D stress and thermal stress

NWDIS2D	Called by DISTBL2D
---------	--------------------

NWDIS3D	Called by DISTBL3D
---------	--------------------

NWDIS2D and NWDIS3D evaluate the displacements at each new node.

NWTMP2D	Called by NUGNUQGR
---------	--------------------

NWTMP3D	Called by NUGNUQGR
---------	--------------------

NWTMP2D and NWTMP3D evaluate the temperature at each new node.

POINTS2D	Called by CORTBL2D
----------	--------------------

POINTS3D	Called by CORTBL3D
----------	--------------------

POINTS2D and POINTS3D evaluate the co-ordinates of the new node

STRSPL	Called by CONOUT
--------	------------------

Special calculations and writes to STR_DATA have been added to distribute the Gauss point stresses,

strains etc. so they are associated with the subelements.

TMPTBL2D	Called by NUGNUQGR
	Calls ADVAR2D and NWTMP2D
TMPTBL3D	Called by NUGNUQGR
	Calls ADVAR3D and NWTMP3D

TMPTBL2D and TMPTBL3D evaluate the temperatures at new nodes which are created when grading elements are divided into subelements. These routines are used for thermal stress analysis and thermal analysis.

Note: Code changes were also made to STRPOST, the stress postprocessor. The following routines were affected: STRPOST, GNURD, GAUSTR, NUGNUQ, STRNOD

Appendix IV: Description of the Adaptive Mesh-Management Algorithm

Those routines which are part of ASGARD¹ are described first, followed by a description of the routine which mimics the finite element program. The other routines which belong to the adaptive algorithm are presented in alphabetical order. A chart of the structure of the adaptive algorithm is also included.

Adaptive Algorithm

Existing ASGARD Routines

ADVAR3D	Evaluates basis functions for element types 146, 148, 149 The routine finds new co-ordinates, temperatures and displacements.
COMPACT	Converts topology list (LNODS) to occupancy list (LOCNDE) and a compacted list of global node numbers (NODGLB)
POINTS3D	Converts (ξ, η, ζ) co-ordinates to (x, y, z) global values
PTNEW	Opens k-D tree CORTBL
PTGET	Returns data from k-D tree CORTBL for a certain key
PTPUT	Stores data in k-D tree CORTBL at a key

New Routines

MOKFEM: reads the input file (topology and co-ordinates) and finds element level error indicators.

	CALL PTNEW (CORTBL)	Open k-D tree for co-ordinates
	CALL PTNEW (PARTBL)	Open k-D tree for parent retrieval
	READ data	Read data for moving point load, with option for manual error entry
A Loop	CALL RDMSH	Read the topology and co-ordinates

¹ A general purpose, transient, nonlinear finite element package written at Carleton University

-PTNEW(PARTBL)

-RDMSH

-ERRCAL

-CNTRL

-INITLZ

-RDMSH

-ERRMSH

-CONNECT

-RFINMSH

-COMPACT

-TOPOL

-ADVAR3D

-POINTS3D

-PTGET(CORTBL)

-PTPUT(CORTBL)

-DVIDE3D

-COMPACT

-TOPOL

-ADVAR3D

-POINTS3D

-PTGET(CORTBL)

-PTPUT(CORTBL)

-CNTRD

-ADVAR3D

-POINTS3D

-PTPUT(PARTBL)

-INITLZ

-CONNECT

-CSNMSH

-COMPACT

-CNTRD

-ADVAR3D

-POINTS3D

-PTGET(PARTBL)

-WRTMSH

1. Loop over increments.
2. Loop over error indicators.
3. Loop over elements for refinement.
4. Forward recursive loop defines propagation path.
5. Backwards loop over propagation path.
6. Loop over edges, surface couples and surfaces adding nodes to neighbours.
7. Loop over local positions 9 to 27 adding nodes to parent.
8. Loop over 8 children.
9. Backwards loop over elements for coarsening.
10. Loop over edges and surfaces removing redundant nodes.

Structure of the adaptive mesh-management algorithm

<i>over</i>	CALL ERRCAL	Get error indicators for the mesh
<i>increments</i>	WRITE errors	Write the error indicator to a file
	CALL CNTRL	Begin mesh management for increment
<i>A ends</i>		Send file names and k-D tree identities

CNTRD: finds the centroid of an element.

CALL ADVAR3D	Evaluate the basis functions at any (ξ, η, ζ)
CALL POINTS3D	Convert (ξ, η, ζ) co-ordinates to (x, y, z) global values

CNTRL: controls the refinement and coarsening loops.

<i>A loop</i>	CALL INITLZ	Initialize arrays (NAYBORS,MIDEDG,MIDSUR)
	CALL RDMSH	Read current mesh
	CALL ERRMSH	Read current errors
	CALL CONECT	Fill arrays (NAYBORS,MIDEDG,MIDSUR)
	For refinement (NORFN > 0)	
	<i>B loop</i>	If ERR(NEL) > 0 then
	<i>over elements</i>	CALL RFNMSH Mesh changes on refinement,
	<i>B ends</i>	therefore leave loop
	For coarsening (NOCSN > 0 and NORFN = 0)	
	<i>C backwards</i>	If ERR(NEL) < 0 then
	<i>loop over</i>	CALL CSNMSH Mesh might change, check
	<i>elements</i>	If change = true
		If (CHANGE) then Leave loop
		If ELEMENT is 1 and NOT (CHANGE)
		then COMPLETE If no changes by element = 1
		coarsening is complete
	CALL WRTMSH	Write the mesh after refinement or coarsening
	If (NORFN or NOCSN > 0) then	
	Go to INITLZ	Loop again for more changes
<i>A ends</i>	Else RETURN	

CONECT fills principal arrays (NAYBORS, MIDEDG, MIDSUR).

<i>A loop</i>	NAYBORS(I,J)	For corner node I
		J = up to 8 elements
		from LNODS(1-8,NEL)

over elements

MIDEDG(I,J)

For midedge node I

J = up to 4 elements

from LNODS(9-20,NEL)

A ends

MIDSUR(I,J)

For midsurface node I

J = 1 or 2 elements

from LNODS(21-26,NEL)

CSNMSH: coarsens a group of siblings.

CALL COMPACT

Get element information

CALL CNTRD

Get the key for parent retrieval
(centroid)

CALL PTGET (PARTBL)

Retrieve parent topology
and common nodeCHECK rules, if
satisfied, continue
else, return

1. Do not coarsen beyond root mesh
 2. Coarsening cannot propagate
 3. All siblings have 8 nodes
 4. All siblings have negative indicators
- If sibling rivalry, reset indicators = 0

LABEL parent

Parent = lowest sibling

TOPOLOGY of parent

Given by retrieval

INCREMENT error indicator

Parent from that of children

TAG siblings for removal

*A loop
over 12
edges of
parent*

CHECK rules

1. Keep a midedge node which
is also a corner node
2. Keep a midedge node which
is holding curvature

REMOVE nodes from neighbours

REMOVE nodes from co-ordinate array

A ends

REMOVE nodes from retrieved parent

*B loop
over 6
surfaces of
parent*

CHECK rules

1. Keep a midsurface node which
is also a corner node
2. Keep a midsurface node which is
holding curvature

REMOVE nodes from neighbours

REMOVE nodes from co-ordinate array

*B ends*REMOVE nodes from retrieved parent

DVIDE3D: divides an element into 8 children (siblings).

	CALL COMPACT	Get element information
	Partially nodal list NUNDE	Filled from LNODS
<i>A loop</i>	Does node EXISTS	check LNODS
<i>over nodes</i>	If not CALL TOPOL	Use (ξ, η, ζ) of position
<i>in positions</i>		Return with nodal identity
9-27	Add node to NUNDE	
<i>A ends</i>	(filled from 1-27)	
<i>B loop</i>	CREATE 8 children	Use NUNDE and data block
	LABEL child	child = largest + 1
<i>over 8</i>	DECREMENT indicator	child = parent - 1
<i>children</i>	CALL CNTRD	Get centroid of each child
	CALL PTPUT (PARTBL)	Place full parent topology in
		k-D tree, node 27 common
<i>B ends</i>		to 8 siblings

INITLZ: initializes critical arrays.

NAYBORS(I,J) = 0	For vertex node I and J=1-8
MIDEDG(I,J) = 0	For midedge node I and J=1-4
MIDSUR(I,J) = 0	For midsurface node I and J=1-2

ERRMSH: initializes, then reads and fills vector of error indicators.

ERR(NEL) = 0	Set vector to 0
READ NOCHN	Get the number of changes, number of refines and coarsens
READ ERR(NEL) = nonzero	Fill with nonzero errors

RDMSH: reads the topology and co-ordinates of a mesh.

READ NELZ, TITLEN	Read number of elements and number of title lines
READ topology	Read element numbers, types, properties and nodes
READ NCOORD	Read node numbers and global (x, y, z) co-ordinates
Fill LNODZG	True number of nodes per element

RFNMSH: for a given element, finds all elements in the propagation path, and then refines them.

<i>A loop</i>	FORWARD recursive	If corner nodes are also
<i>using</i>	loop determines	midedge or midsurface nodes

<i>elements</i>	propagation paths	in a neighbour, propagate
<i>A ends</i>	ELDIV(NEL)	Holds propagation path
<i>B loop</i>	BACKWARD loop over ELDIV(NEL)	
	Need 12 lists of edges	Use corner nodes
	Need 6 lists of surfaces	Use only corner nodes
<i>C loop</i>		For each node on edge find element neighbours
<i>over 12</i>		For each contacting edge in a neighbour identify
<i>edges</i>		the corner nodes between which
<i>C ends</i>		a node may need to be added
<i>D loop</i>		For each node on surface couple find element neighbours
<i>over 12</i>		For each contacting edge in a neighbour identify
<i>surface couples</i>		the corner nodes between which
<i>D ends</i>		a node may need to be added
<i>E loop</i>		For each node on surface find element neighbours
<i>over 6</i>		For each contacting surface in a neighbour identify
<i>surfaces</i>		the corner nodes between which
<i>E ends</i>		a node may need to be added
<i>F loop over</i>		Get local nodal positions
<i>12 edges x 3</i>		of corner nodes on edges
<i>F ends</i>		in up to 3 neighbours
<i>G loop over</i>		Get local nodal positions
<i>12 surface</i>		of corner nodes on edges
<i>couples x 2</i>		in up to 2 neighbours
<i>G ends</i>		
<i>H loop over</i>		Get local nodal positions
<i>6 surfaces</i>		of corner nodes on surfaces
<i>H ends</i>		in 1 neighbour
<i>I loop</i>		Get (ξ, η, ζ)
<i>over 12</i>		of midedge nodes
<i>edges x 3</i>		in up to 3 neighbours
		(ξ, η, ζ) defines local nodal position
	Does node EXIST?	Check LNODS
	If not CALL COMPACT	Get information for neighbour
	CALL TOPOL	Get nodal identity

<i>I ends</i>	Place node in neighbour
<i>J loop</i>	Get (ξ, η, ζ)
<i>over 12</i>	of midedge nodes
<i>surface</i>	in up to 2 neighbours
<i>couples x 2</i>	(ξ, η, ζ) defines local nodal position
	Does node EXIST? Check LNODS
	If not CALL COMPACT Get information for neighbour
	CALL TOPOL Get nodal identity
<i>J ends</i>	Place node in neighbour
<i>K loop</i>	Get (ξ, η, ζ)
<i>over 6</i>	of midsurface nodes
<i>surfaces</i>	in 1 neighbour
	(ξ, η, ζ) defines local nodal position
	Does node EXIST? Check LNODS
	If not CALL COMPACT Get information for neighbour
	CALL TOPOL Get nodal identity
<i>K ends</i>	Place node in neighbour
CALL DVIDE3D	All neighbours have required nodes
	Divide the parent element
CALL INITLZ	Initialize critical arrays
CALL CONECT	Fill NAYBORS, MIDEDEG, MIDSUR
<i>A ends</i>	Skip for last element in path

TOPOL: checks for the existence of a certain node.

CALL ADVAR3D	Evaluate the basis functions for (ξ, η, ζ)
CALL POINTS3D	Get global co-ordinates of the point
CALL PTGET	Global co-ordinates are the key, does the node exist?
CALL PTPUT	If not, store the nodal identity at the key
	Add the node to the list of co-ordinates

WRTMSH: overwrites the existing elements and nodes.

WRITE NELZ, TITLEN	Number of existing elements and title size
WRITE LNODS	For each element, its type, property, and nodes
WRITE NCOORD	For each node, its number and global co-ordinates
WRITE NOCHN	The number of elements to be changed
WRITE NEL, ERR(NEL)	Elements and their nonzero errors

Appendix V: Description of Arbitrary Coarsening

A brief map of the subroutine calls is presented first. The map ends at the call to CNTRL because routines after CNTRL, are identical to those for the adaptive mesh-management algorithm. Please see Appendix IV for the description of these subroutines. Only the subroutines which were created specifically for arbitrary coarsening are discussed below.

HUNTER

- PTNEW (CORTBL)
- PTNEW (PARTBL)
- RDMSH
- INITLZ
- CONECT
- ADOPT

- ADOPT2
- ADOPT4
- ADOPT8
- COMPACT
- CNTRD
- PTPUT

- CNTRL — see Appendix IV

ADOPT: determines the number of children to be adopted based on their common node. Each group has subsets and other adoption routines are called accordingly.

<i>A loop</i>	FIND siblings	Use the common node to extract the
<i>over up to</i>		number of siblings using NAYBORS
<i>8 nodes</i>	CHECK	All siblings must have 8 nodes
<i>A ends</i>		There must be 2, 4 or 8 neighbours
	CALL ADOPT2 or	Create parent adopting 2 children
	CALL ADOPT4 or	Create parent adopting 4 children
	CALL ADOPT8	Create parent adopting 8 children
<i>B loop</i>		SET-ERR(sibling) = -1,
<i>over</i>		CALL COMPACT Get element information
<i>up to 8</i>		CALL CNTRD Get centroid
<i>siblings</i>		CALL PTPUT Place adopting parent topology
<i>B ends</i>		in k-D tree PARTBL

ADOPT2: adopts the parent element for 2 neighbours which share a common node and a common face.

<i>A loop</i>	Do surfaces match	Cases: 3 ways of
<i>over corner</i>		connecting 2 elements at
<i>nodes of 2</i>		a shared surface; on x-y,
<i>neighbours</i>		x-z, or y-z planes, each has
<i>A loop ends</i>		2 subsets of node numbering
<i>B loop</i>	Identify CASE	The surfaces match,
<i>over 6 surfaces</i>		the counter value gives CASE
<i>with 4 nodes</i>		from 6 possibilities
<i>B loop ends</i>		for parent numbering

For CASE get NODLST from LNODS of siblings

ADOPT4: adopts the parent element for 4 neighbours which share a common node and a common edge.

<i>A loop</i>	Do edges match	Cases: 3 ways of
<i>over corner</i>		connecting 4 elements at

nodes of 4
neighbours
A-loop ends

a shared edge; in x, y,
or z

Identify CASE

Use the edge in sibling 1,
get difference in x co-ordinates
y co-ordinates and z co-ordinates
use NCOORD and LNODS
1 of them is nonzero and gives CASE

RANK 4 siblings for order of assembly

For CASE get NODLST from LNODS of siblings using RANK

ADOPT8: adopts the parent element for 8 neighbours which share a common node.

A loop
over 8 corner
nodes of 8
neighbours
A loop end

COMMON NODE is

matching node
for all the
neighbours

There is 1 way of
connecting 8 elements at
a shared point

RANK the 8 siblings for order of assembly

For CASE get NODLST from LNODS of siblings using RANK

HUNTER: helps the user search for the area to be coarsened. It then calls the adoption routines.

CALL PTNEW (CORTBL)

Initialize k-D tree for co-ordinates

CALL PTNEW (PARTBL)

Initialize k-D tree for parent retrieval

READ file names

User is requested to give file names

CALL RDMSH

Read the topology and co-ordinates

CALL INITLZ

Initialize critical arrays

CALL CONECT

Fill critical arrays (NAYBORS, MDEDG, MIDSUR)

INTERACTIVE

Interactive discussion with user identifies areas
of interest in the mesh.

Both refinement and coarsening are possible

If refinement occurs the unique parent-child
relationship is stored.

CALL CNTRL

Begin the regular routines for mesh management.

END

08·08·88

FIN