Arbitrary 2-Element H-Coarsening of a TTH-Generated Hexahedral FEA Mesh

By

Sándor D. Kostya, B.Eng.

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of

Master of Applied Science

Department of Mechanical and Aerospace Engineering
Ottawa-Carleton Institute of
Mechanical and Aerospace Engineering

Carleton University
Ottawa, Ontario
January 2005

Your file Votre référence
ISBN: 0-494-06794-2
Our file Notre référence
ISBN: 0-494-06794-2

The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis,

Arbitrary 2-Element H-Coarsening of a TTH-Generated Hexahedral FEA Mesh

Submitted by
Sándor D. Kostya, B.Eng.
in partial fulfilment of the requirements for the degree of
Master of Applied Science

_____
J. M. J. McDill, Thesis Supervisor

_____
J. Beddoes, Chair, Department of Mechanical and Aerospace Engineering

Carleton University
January 2005

ABSTRACT

The development of finite element hexahedral element generation has been an ongoing process with progress concentrated in areas of robust, autonomous mesh generation. Recently, an application for converting complex geometries from tetrahedral meshes to fully hexahedral meshes (TTH) has been developed. The TTH application uses a dicing method to split each tetrahedron into four hexahedra. This conversion software has been somewhat problematic in that it creates non-orthogonal hexahedral meshes that are overly refined in non-critical areas. Therefore, a need for selectively coarsening unstructured irregular TTH-generated meshes has been identified.

The development and testing of a selective coarsening application is presented along with the Graphic user Interface to TTH (GITTH). The GITTH application provides a user-friendly interface, which ties the TTH program together with its peripheral software, as well as provides a method for selective coarsening of hexahedral meshes. The coarsening algorithm uses an 8- to 26-node linear hexahedron to support mesh grading and utilizes arbitrary coarsening principles to match elements for coarsening.

The coarsening algorithm supports coarsening 2 sibling elements into a single parent element at user-selected areas within a mesh. This algorithm has been found capable of coarsening various orthogonal and non-orthogonal, irregular meshes. Although, the coarsening routine does not improve local mesh quality, it is able to detect and avoid creation of severely distorted elements in the area of coarsening.

ACKNOWLEDGEMENTS

I would like to express my gratitude to all the people who have supported me throughout this project:

- To Professor McDill, whose assistance made it possible for me to undertake this project. Her guidance and support will always be appreciated.
- To my loving wife Karra, who has stood with me throughout my graduate studies and has always pushed me to succeed. Her patience and understanding helped my transition back into academic life.
- To my parents, for their unconditional support and encouragement in my decision to return to school to pursue graduate studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

b, h	Width and height of a hexahedron
det J	Determinant of the Jacobian
Det $J_{0,0,0}$	Determinant of the Jacobian at the centroid
{f}	Vector of normal forces
g	Metric tensor
g_{ij}	Metric tensor components
[J]	Jacobian matrix
J^T	Transpose of the Jacobian
[K]	Stiffness matrix
L	Characteristic dimension
(min. det $J)_{r,s,t}$	Minimum determinant of the Jacobian found at computational coordinates r,s,t
NE	Number of neighbouring elements connected to a node
{Q}	Vector of heat fluxes
{T}	Vector of temperatures
{u}	Vector of displacements
r,s,t	Coordinates in computational domain
w	Weight factor in isoparametric smoothing
x,y,z	Coordinates in physical domain
X, Y, Z	Global coordinates in physical domain
x'	Obtained coordinate value after isoparametric smoothing
x_A, x_B, x_C	Adjacent node coordinate to the intended node for smoothing
x_D	Element node that is in the opposite corner to the intended node for smoothing
θ_{ij}	skew angles between r, s, t computational axis in physical space

AR	Aspect Ratio
CFD	Computational Fluid Dynamics
DP	Distortion Parameter

FEA	Finite Element Analysis
FEM	Finite Element Method
GITTH	Graphical Interface to TTH
TTH	"Tetrahedra-to-Hexahedra" conversion

Chapter 1: INTRODUCTION

Finite element analysis (FEA) has become an integral tool used within many engineering fields today. It provides engineers with a useful method for analyzing designs where classical methods have not been able to give accurate, detailed results. This is especially seen in cases where irregular, complicated geometries or non-homogenous materials are required. FEA applications have become increasingly used within design communities and have become mainstays in many product development groups. New products and future developments are often driven by the results obtained from FEA [1].

Commercial software packages have taken FEA from a mainly structural analysis tool to one that handles heat transfer, fluid flow, electrical, magnetic, crack propagation, acoustic and dynamic problems, to name a few [1][2]. Although the mathematical principles for each problem type may vary, they all employ the same finite element method (FEM) to solve these problems.

FEM divides a user-defined geometry into smaller finite element segments. The geometry can be two or three dimensions, depending on the problem. Once the geometry has been discretized (meshed), each element is coupled to its neighbours through an algebraic set of equations in matrix form, and solved.

In the past, computational processing was responsible for the greater percentage of time and money spent on performing an analysis. Now, with increased computational power and the availability of inexpensive equipment, pre- and post-processing (e.g. meshing operations) are taking up a greater amount of the time and resources required to perform an analysis [1].

To reduce the time required to generate good quality meshes, much of the pre-processing development have been focused on creating robust, autonomous mesh generation software. Currently, autonomous tetrahedral element mesh generation has been developed and proven to work, but autonomous hexahedral element mesh generation has not had the same degree of success [3].

Most commercial FEA packages include tetrahedral element (freemeshing) generation options, but their hexahedral element mesh generation options are limited to mapped meshing [1]. Tetrahedral element meshes have been found to be much easier to generate than their hexahedral element counterparts. Although easier to generate, tetrahedrons may produce less accurate results than those simulations using hexahedrons [4]. For this reason, many nonlinear computational simulations are performed using hexahedral meshes [5].

Ongoing research into hexahedral element generation has led to significant improvements in this field. Recently, an application for converting complex geometries from tetrahedral meshes to fully hexahedral meshes (TTH) has been developed [6]. The TTH application uses a dicing technique to split each tetrahedron into four hexahedra. This conversion software has experienced some challenges in that it creates non-orthogonal hexahedral meshes that are overly refined in non-critical areas. Therefore, a need for selectively coarsening unstructured, non-orthogonal TTH-generated meshes has been identified to reduce the computational costs associated with large meshes.

This thesis presents the design, development and validation testing of an algorithm to selectively coarsen TTH-generated meshes. The algorithm has been designed to work with TTH-generated meshes specifically utilizing the mesh format of

TIAMAT[1]. To link the TTH software to the coarsening scheme, a graphical user interface, named GITTH (GUI Interface to TTH), was developed.

Chapter 2 of this thesis briefly reviews the background to finite element analysis and presents meshing and mesh-management schemes found in literature. Chapter 3 reviews the tetrahedra-to-hexahedra (TTH) conversion method and discusses the challenges with coarsening such meshes. Chapter 4 outlines the coarsening algorithm developed to handle TTH meshes and Chapter 5 gives a description of the GITTH application. Chapter 6 presents results and observations made during the performance and verification testing. The performance of the coarsening algorithm is discussed in Chapter 7 and the conclusions and recommendations for future work are presented within Chapter 8.

[1] TIAMAT is a multipurpose FEA code developed at Carleton University, for the analysis of 3-D thermal-mechanical problems.

Chapter 2: FINITE ELEMENT MESH BACKGROUND

The finite element method (FEM), also known as finite element analysis (FEA), has become a powerful tool in the analysis of engineering problems. FEA is one of the main tools engineers use to simulate and analyze designs in varying fields of engineering. Model discretization (meshing) is one piece of the FEA process, and is the main focus of this thesis.

To provide some background of how meshing relates to finite element analysis, a general description of FEA is introduced below. This will be followed by a description of meshes and methods for generating and managing meshes.

2.1 Finite Element Analysis Background

The finite element method is a numerical approach for analyzing engineering problems. Although the mathematical foundation on which FEA was built has been under development since the 1800s, it is generally agreed that the first early forms were utilized during the 1940s [1][2].

Over the last few decades, FEA tools have evolved to tackle problems in various engineering fields (i.e. stress analysis, fluid dynamics, heat transfer, etc.) [2]. Current FEA tools are comprised of three main components: a pre-processor, a processor and a post-processor.

2.1.1 Pre-Processor Stage

The model discretization (meshing) occurs during the pre-processor stage. Typical commercial pre-processor software gives a user the ability to select the method

for meshing an object as well as some control over the mesh density and element quality. It is also during this stage that material properties and boundary conditions are applied. All of this information is important because it defines the problem in a format that the processor uses for analysis.

2.1.2 Processor or Solver Stage

The second stage is the processor or "solver". In this stage, the numerical approximate solution to the problem as it was defined within the pre-processor stage is calculated. The solver uses the material properties and topology, defined in the pre-processor stage, to assemble a global stiffness matrix [K]. The stiffness matrix and the applied boundary conditions are then used to solve a system of global algebraic equations.

In the case of a static structural analysis, the boundary conditions are applied nodal forces (f) and nodal displacements (u). The linear elastic case reduces to:

$$[K]\{u\} = \{f\} \tag{2.1}$$

In the case of a steady-state thermal analysis, the boundary conditions are nodal heat fluxes (Q) and nodal temperatures (T). For steady-state heat transfer the equation reduces to:

$$[K]\{T\} = \{Q\} \tag{2.2}$$

Processors are typically categorized as either implicit or explicit, although some processors combine both implicit and explicit schemes into a hybrid solver [7][8]. Each solver scheme has its own advantages and limitations and each is generally better suited for specific types of problems [9].

An implicit solver attempts to advance a solution by solving[2] a system of equations for each incremental time step, from a starting state until the solution converges at the final time step. Implicit algorithms tend to be numerically stable and allow large time steps. This type of solver is most beneficial when solving steady-state or quasi steady-state problems where linear equations are readily formed and solved in an efficient manner. With nonlinear, large deformation problems, implicit solvers are found to be less efficient. This inefficiency is due to their iterative nature, where the computational cost can become very large since the system of equations must converge to a solution for each time step [7] [9][10].

An explicit solver determines a solution without the need for iteratively solving a large system of equations. It does this through direct time integration. This type of solver has a strong advantage over implicit methods when solving nonlinear problems because it requires far less computational power. However, while explicit solvers avoid iterative equation solving, the time-step sizes are restricted to ensure numerical stability of the solution [7].

2.1.3 Post-Processor Stage

The post-processor is the third stage in the FEA process. This stage displays the results from the processor in a format that easily interpreted by a user.

Most post-processors are capable of displaying the solution data through animations, colour fringe plots, contour plots, or vector plots. From the data displayed, a user can assess the results, and determine if further effort is required to improve the

[2] Implicit solvers generally use either direct or indirect methods to solve a system of equations. Direct methods use a form of Gaussian elimination to solve a set of equations. Indirect methods use techniques such as the element-to-element conjugate gradient scheme to iteratively solve a system of equations.

quality of the analysis. If improvements to the analysis are needed, an analyst may go back to the pre-processor stage and refine the problem by changing the mesh density, changing element types and/or changing the boundary conditions.

2.2 Discretization

FEM works on the principle of sub-dividing complex geometries into a finite number of small pieces, called elements. By producing elements of a simple shape, the problem is broken down into a set of manageable partial differential equations. Each element is represented within the global set of equations, which describes their connectivity to neighbouring elements.

A finite element mesh data set contains all connectivity and topology information of a discretized 2-dimensional (2-D) or 3-dimensional (3-D) geometry. This information comes in the form of local element composition and nodal coordinates. Elements use the nodal information to define their shape and connectivity to surrounding elements. Nodes are points within the geometry consisting of the geometric (x, y, z) coordinate information.

The size, shape and polynomial order of elements within a mesh, all play important roles in the accuracy and efficiency of an analysis. The ways in which these parameters influence an analysis are described below. Also, details on meshing schemes and element types are presented with the focus on hexahedral elements.

2.2.1 Structured Versus Unstructured Meshes

Meshes are typically characterized as structured or unstructured meshes. Structured meshes are grid based or have rectangular shaped external boundaries. Within

a structured mesh, all interior nodes are adjacent to an equal number of elements. Each element can be associated directly with ordered indices where each neighbouring cell indices differ by a fixed interval [10]. These meshes are most commonly used in the area of computation fluid dynamics (CFD), where strict grid based alignment of elements may be required by the analysis software to acquire an accurate result [3]. Figure 2.1 gives two examples of structured meshes.

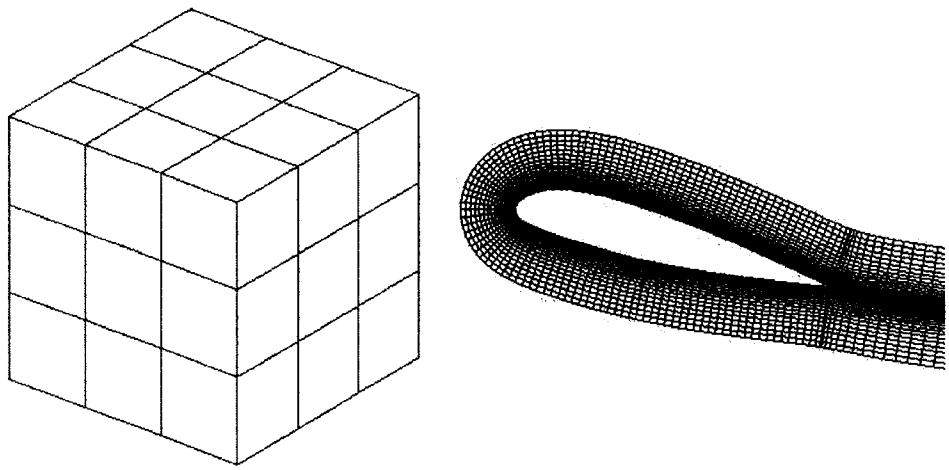


Figure 2.1: Structured mesh examples. (a) 3-D (b) 2-D [11]

Unstructured meshes do not require a blocky or rectangular domain for generation. Numerous unstructured algorithms have been developed that give greater flexibility to generating meshes over complex geometries. This greater flexibility comes in part, from allowing the number (valence) of neighbouring elements per interior node to vary [3].

Many of these mesh algorithms can be automated, as seen with Delaunay triangulation techniques, which create tetrahedral (3-D) or triangular (2-D) element types.

Unstructured meshes have the advantage that they can mesh irregular, non-rectangular geometries and allow mesh refinement in local topological areas. The disadvantage of unstructured meshes lies within the manner in which the mesh data is stored. All element connectivity and nodal topology must be stored separately. This means extra computational resources are required for storing and handling unstructured mesh data when compared with structured meshes [10]. Figure 2.2 gives two examples of unstructured meshes.

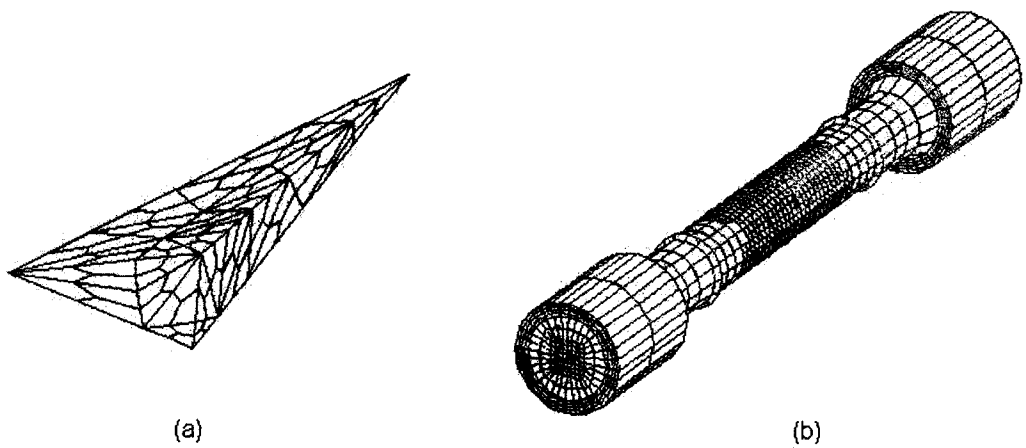


Figure 2.2: Unstructured mesh examples [6][12].

2.2.2 Element Types

There are a number of element types used in FEA simulations. They fall under the category of either continuum elements or structural elements.

Continuum elements can be 2-D or 3-D and their geometry is completely defined by their nodal coordinates [13]. Quadrilaterals and triangles are the most commonly used elements in 2-D simulations. Three-dimensional simulations use tetrahedra (tet) and

hexahedra (hex or brick) elements as a mainstay, while pentahedra (wedge or pyramid) elements are sometimes used to connect hex and tet elements. Figure 2.3 shows the differences between the element types mentioned above.

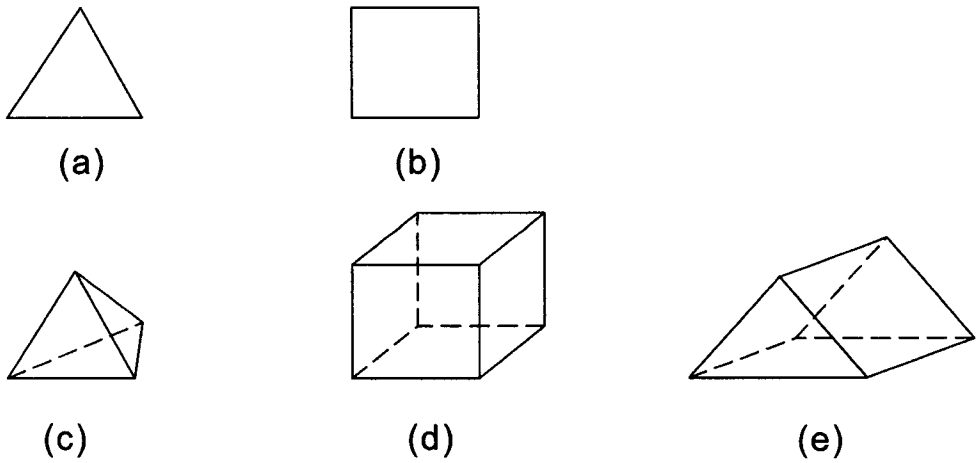


Figure 2.3: Continuum elements. (a) Triangle element (b) Quadrilateral element (c) Tetrahedral element (d) Hexahedral element (e) Pentahedral element

Structural elements have been designed to behave according to structural models found in solid mechanics theory. Their geometry is not completely defined by their nodal coordinates so a user must add extra information during mesh generation.

Beam and shell elements are the most common within this category. It is worth noting that some FEA applications include specialty elements such as spring, damper, mass, rigid, gap and contact elements to help facilitate complex boundary conditions and idealize assembly interfaces [1].

2.2.3 Isoparametric Elements

Elements are considered to be isoparametric, when they use the same interpolation (basis) functions to define the geometry and parameter functions (i.e.

displacements for stress analysis) at each node [2][13]. Isoparametric elements make it possible for an element to have curved edges and makes mesh-grading (local refinement/coarsening) much more efficient [14].

Isoparametric hexahedrons use basis functions to interpolate parameter functions, such as displacements or temperatures, between the nodes of an element. Further, the basis functions also govern element geometry since they are used to map elements in physical space (x, y, z coordinates) to an ideal, perfectly orthogonal brick element in computational space (r, s, t coordinates).

Figure 2.4 depicts an 8- to 26-node isoparametric linear hexahedron element proposed by McDill et al. [15]. It is an extension of the familiar 8-node linear isoparametric brick where the midedge and midface nodes are optional, depending on the mesh grading surrounding the element. It can also be nonconforming through the use of nodeless bending modes [16].

The work presented in this thesis uses the 8- to 26-node brick because it permits an efficient means for mesh grading.

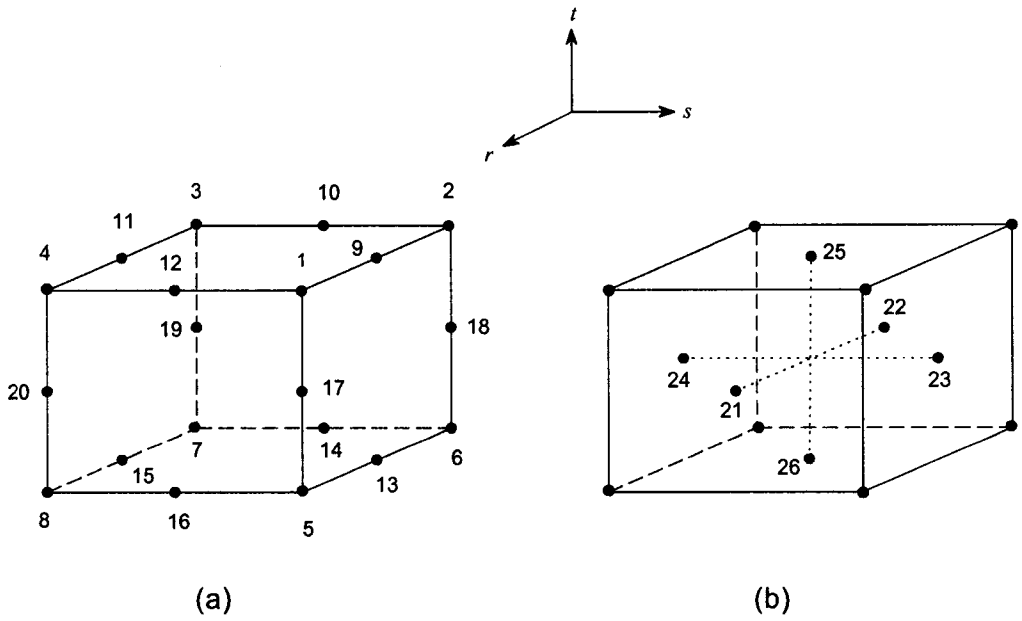


Figure 2.4: A 26-node isoparametric hexahedron [15]. (a) Corner and optional midedge nodes (b) optional midface nodes

2.2.4 Mesh Generation Methods

Mesh generation techniques are a constantly evolving technology. They have advanced due to the need for creating well-shaped elements while reducing meshing times. Originally, meshes had to be manually created and fit within a desired geometry. This was both difficult and time consuming.

With meshing advances, techniques have been developed to automatically mesh geometries. This has helped considerably to speed the time it takes to mesh geometry. Unfortunately, these techniques are mainly limited to triangle or tetrahedral elements.

Most current commercial packages have the ability to automesh 2-D and 3-D geometries with triangles, quadrilaterals or tetrahedrons [1]. There does not yet appear to be an automatic method for creating well-shaped hexahedrons with the same consistency

over a range of geometry shapes as is performed by tetrahedrons. Although not fully automatic, there are some limited hexahedral meshing algorithms available to mesh specific geometries. These algorithms are described in the sections below.

2.2.4.1(2-D) Triangle/(3-D) Tetrahedral Meshing

Automeshing with triangles or tetrahedrons (also called freemeshing) is the most common method used for meshing geometries. This is due to the relative ease and speed in which these techniques work. Whether the geometry is 2-D or 3-D, most freemeshing techniques fall into one of three categories: 1) Octree, 2) Delaunay, 3) Advancing Front [3]. Since the work presented within this paper focuses on hexahedral elements, the following description of these categories will be cursory.

The Octree method recursively subdivides a geometric entity into cubic sections until a desired resolution is reached. To decompose the geometry in such a manner, many surface intersection calculations are necessary to find where the cubic grid meets the geometry's surface. Irregular cubes are created near the surface when the geometry is not rectangular in shape. Triangles or tetrahedrons are generated out of the regular and irregular cubic sections [3][17]. Figure 2.5 demonstrates this method as depicted through a quadtree (2-D) representation.

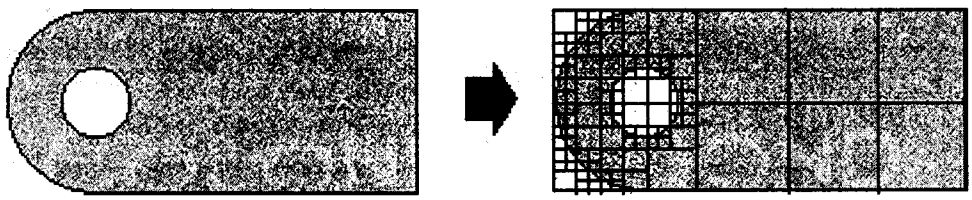


Figure 2.5: Quadtree subdivision of a 2-D geometry [3].

The Delaunay criterion is perhaps, the most widely used technique used for freemeshing [3]. This method is not an algorithm for creating elements, but instead provides criterion for which to connect nodes within a volume. The criterion states:

> "any node must not be contained within the circumsphere of any tetrahedral within the mesh. A circumsphere can be defined as the sphere passing through all four vertices of a tetrahedron" [3].

Although this technique provides a method for connecting nodes into an element, it does not create the nodes. A method for generating and placing nodes is still required.

Advancing front algorithms recursively build elements from the surface inward. Once a layer of elements is created at the surface, the front advances inward and starts another layer of elements that is built based on the previous layer. These types of schemes must have an intersection check to make sure elements do not overlap as different fronts advance toward one another. Figure 2.6 gives an illustration of the advancing front scheme.

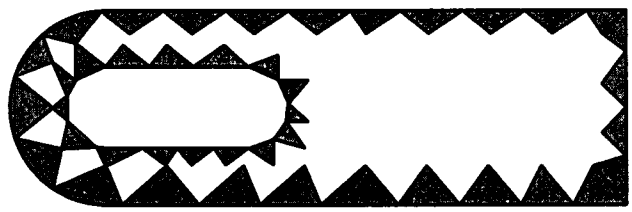


Figure 2.6: Advancing front scheme where the first layer of elements is applied to a 2-D geometry [3].

2.2.4.2 (2-D) Quadrilateral/(3-D) Hexahedral Meshing

While tetrahedra and triangle elements are more commonly used, they are not always the best choice of elements. Hexahedral and quadrilateral elements are more

difficult to generate, but they have been found to produce better results [1]. Benzley et al. [4] compared linear and nonlinear FEA test cases using all tetrahedral and all hexahedral meshes. Both linear and higher order elements were used for testing each element type. Benzley's evaluation demonstrated that hexahedral elements give more accurate results than those of tetrahedral elements. This can be attributed to their ability to deform at a lower strain energy state [4]. Higher order tetrahedral elements are considered acceptable for most linear problems, but hexahedrons are typically required for nonlinear cases [1][4].

The generation of 3-D hexahedral meshes has been found to be more problematic than generation of 2-D quadrilateral meshes. Unlike triangle/tetrahedra generation, it has not been as easy to expand 2-D quadrilateral algorithms into reliable 3-D hexahedral algorithms. The complexity involved in generating such elements is the reason why there is not a fully autonomous hexahedral mesh generator on the market.[3]

Algorithms that generate hexahedral meshes can be categorized into two main approaches: direct and indirect schemes.

Direct Schemes

The majority of meshing tools on the market use direct schemes. The likely cause of this is due to the low quality of element shapes, created by indirect methods. Direct algorithms build a mesh directly from the geometry supplied. These types of schemes use the initial geometry surfaces and build hexahedrons to fill the volume. There are four separate approaches seen in unstructured, solid, all-hexahedra meshing literature [3]:

[3] Recent commercial meshing applications, such as TrueGrid (distributed by XYZ Scientific Applications Inc.) have the ability to mesh a large variety of geometries with hexahedral elements. It is unknown to the Author at this time as to the limitations of these applications.

1. Grid-Based
2. Mapped Meshing
3. Plastering
4. Whisker Weaving

Grid-Based

The grid-based approach superimposes a cube shaped grid of a desired density, onto a solid geometry. The grid is first implemented within the interior of the model to create hexahedral elements and then fills the surface boundary layer with hexahedral elements fit to match the surface (as depicted in Figure 2.7). This approach is effective for creating well-shaped elements in the interior, but has difficulty creating elements of the same quality at the boundary [3][18]. In addition, the elements at the surface boundary are dependent on the orientation of the grid and may not be well aligned to the boundary [3].

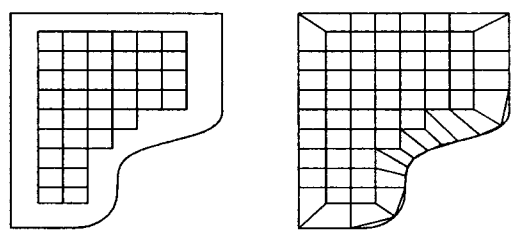


Figure 2.7: Grid-based mesh generation of a 2-D geometry [18]. (a) Interior element creation (b) Surface boundary element creation

Recently, Su et al. [19] has modified this technique to generate a mesh over multiple domains, using a hybrid node projection algorithm. To date, this scheme has experienced problems with boundary insensitivity. Boundary insensitivity, within their algorithm, has been found to overly refine elements at the surface boundary.

Mapped Meshing

Mapped meshing is a structured mesh generation technique that uses a perfectly orthogonal hexahedral mesh in computational space and maps it onto a volume in physical space. To create a good fit of the mesh, a form of blending is used to match it to the geometry. The Coons patch is an example of one method used for blending [20][21].

This technique has become widely used for unstructured mesh generation as it creates good quality elements for blocky, rectangular shaped geometries. Figure 2.8 gives a 2-D example of mapped meshing. For mapped meshing to work in a 3-D environment, the geometry must allow the surface mesh, on opposite faces, to have the same number of elements. This typically means a 3-D object must have 6 distinct faces; however, some algorithms can handle objects with a fifth rounded surface (as shown in Figure 2.9). If an object does not have clear opposite faces, this technique is not always able to produce a good quality mesh [3].

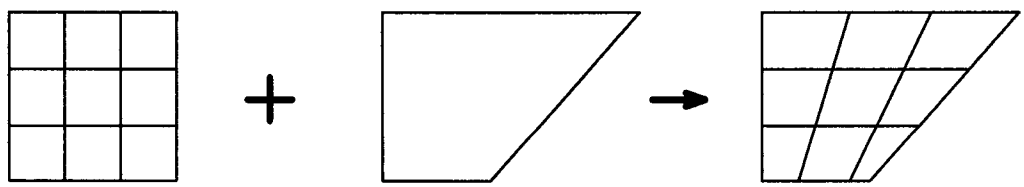


Figure 2.8: Mapping a 2-D quad mesh onto a blocky, rectangular shaped geometry.

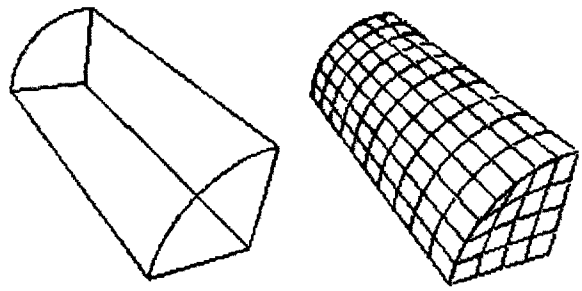


Figure 2.9: Mapped mesh of a 5-sided volume [22].

Because many of the 3-D designs are a mix of geometrical forms, some techniques have been devised to decompose complicated geometries into map-meshable regions. This method of meshing is known as sub-mapping.

Sandia National Laboratories in the U.S., has been developing the CUBIT software [22] to advance hexahedral meshing algorithms. Extensive research has been done using CUBIT to develop an automatic feature recognition algorithm that is capable of decomposing volumes into mappable regions. Once the features of a volume are charted, virtual surfaces are established at each feature boundary. The subsequent regions are then map-meshed separately [3][23]. Submapping has the advantage of automatically creating well-shaped meshes over complex objects, but is limited to blocky volumes with well-defined corners and rectangular sections.

Some volumes are ideal for protruding or sweeping a mesh from one surface to another. This category of mapped meshing is referred to as sweeping or 2.5-D meshing. To categorize a volume as 2.5-D, it must consist of an initial 2-D face, which has been projected through space, along a linear and/or curved path, to a final position. The resulting 3-D object must have a similar cross-section throughout. Examples of this class of geometry are cylinders, springs and piping. The majority of sweeping techniques

involve projecting a surface quad mesh along a trajectory to a target surface. Figure 2.10 gives an example of a swept volume.

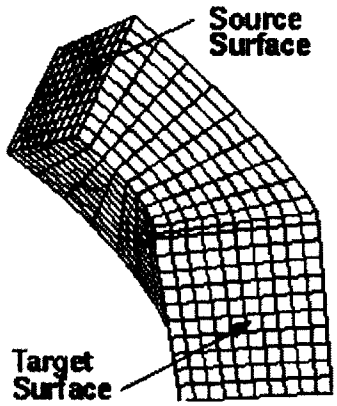


Figure 2.10: Hex elements swept through a volume [24].

Previous sweeping algorithms required a 2.5-D volume and were only capable of sweeping along a single path. However, recent advances made by Blacker [25] and Lai et al. [26] have produced techniques where multiple sources and multiple targets of a single volume are able to be swept concurrently. These schemes are termed 'Multisweep'. Multisweep techniques still require a single sweep direction, except now the mesh path is capable of splitting and branching towards separate targets or starting from separate sources and converging to a single target. Because these schemes loosen the requirements for the source and target face to have similar topology, a greater class of geometries may be swept. Sweeping and multisweeping is limited to certain classes of geometry, but does generally create good quality meshes for volumes in such categories [3][26].

There are two final map meshed schemes discussed in this thesis. The two final map meshes are the medial surface [27] approach and the embedded Voronoi graph [28] approach.

The medial surface scheme involves decomposing a volume into a set of map mesh-able regions using medial surfaces. Medial surfaces are a 3-D extension of the 2-D medial axis. The medial axis is a line created by the centre of a maximal sphere as it rolls through a volume [27]. Using the same principles, the medial surface method creates surfaces along the path of the medial axis. These surfaces define the boundary of each map meshable region.

This medial surface scheme method is not overly robust, and has several drawbacks [3][28]. First, it tends to generate models with overly fine subdivisions. Second, the algorithm used for computing the medial surface is not proven and may create degenerative surfaces. Last, its use of midpoint subdivision can lead to poorly shaped elements.

The embedded Voronoi graph approach decomposes a volume into subvolumes that can be swept. The embedded Voronoi graph contains all the symbolic information of the Voronoi diagram, medial axis and a geometric approximation of the object. This approach appears to be stable and creates good quality meshes, but it does have the drawback of occasionally overly decomposing a volume [28].

Plastering

Plastering is an extension of the paving algorithm (also called the advancing front technique for tetrahedral mesh generation). Paving techniques attempt to mesh arbitrary 2-D geometries or 3-D surfaces with all quadrilateral elements [29][30].

Similar to the advancing front method, the plastering algorithm paves layers of hexahedra along a boundary or front. As a layer of elements is finished, it becomes the

front from which the next layer is built. Layers of elements are recursively created, working inwards until the volume is filled. Figure 2.11 illustrates plastering in a 3-D object.

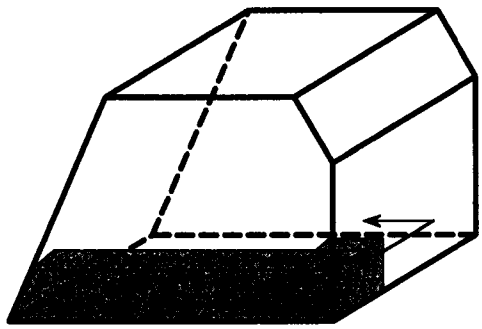


Figure 2.11: Plastering a 3-D object [3].

Plastering starts with quadrilateral elements paved over a surface of volume. The quadrilateral elements are extruded into the volume to form hexahedra along an advancing front. The order of fronts is selected from a set of heuristic procedures based upon geometry. As the fronts advance, complex algorithms are required to detect element intersections and seam creation (small unmeshed areas in the domain which have small angles between the neighbouring fronts). Towards the interior, small intricate voids are known to appear, which cause great difficulties in generating hexahedral elements to fill the space. Many of these voids can only be filled with wedges or tetrahedrons.

Although paving algorithms are proven to be robust, plastering methods for all hexahedral meshes have not proven reliable due to their algorithmic complexity [3].

Whisker Weaving

Whisker weaving is an approach based on the advancing front algorithm, but it uses the spatial twist continuum (STC) concept to construct the connectivity of the mesh. Tautges and Blacker [31] first introduced this technique as an alternative to plastering schemes due to difficulties in implementing a robust plastering algorithm.

This method first builds a quadrilateral surface mesh over the boundary of the volume. Next, it constructs hexahedra element connectivity using a STC scheme integrated with elements of the advancing front technique. The connectivity information does not contain nodal information, only the STC data, which is a global data structure made up of intersecting planes.

The STC is a geometric scheme for breaking a volume down into intersecting planes, known as whisker sheets. Whisker sheets propagate through a volume, bisecting other sheets. The intersection of three sheets creates a vertex, two sheets create a chord and the dangling end of a chord is called a whisker. Figure 2.12 illustrates a simple example of STC within a rectangular object.

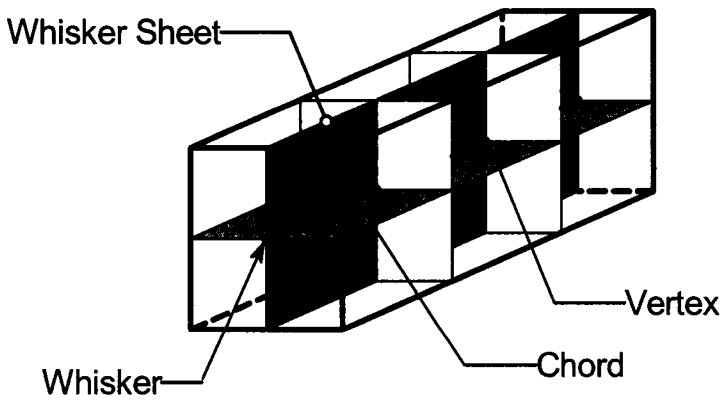


Figure 2.12: A rectangle containing STC information (Whisker Sheet, Vertex, Chord and Whisker) [31].

Hexahedra elements are constructed within the volume once the connectivity has been established. As each hex element is built around an STC vertex, node placement is performed.

The whisker weaving method has shown promise for generating all-hexahedral meshes for large and complex volumes. Despite these positive findings, the method also has a few drawbacks. First, poor quality elements are often constructed in isolated regions of the mesh. Ongoing research is underway to improve this defect. Secondly, degenerate hexahedrons known as knives, are sometimes formed when the mesh consists of an odd number of quads on the surface of the volume [32].

Indirect Schemes

Indirect schemes involve transforming a previously created tetrahedral mesh into a hexahedral mesh. These types of schemes have the advantage of avoiding computationally expensive intersection checks and geometric decomposition routines that are intrinsic to many direct algorithms [33]. They are also considered very fast algorithms, because they only require local topological data from the initial tetrahedra mesh to generate a hexahedron [3]. The greatest advantage of these schemes is their ability to mesh complex, irregular objects with hexahedral elements.

The main drawback with using indirect methods is that they generally create irregular nodes. An irregular node on the interior of a hexahedral mesh is defined as any node that is not adjacent to exactly eight elements. Instead, they may have more or less than eight neighbouring elements. These types of nodes are a drawback because they tend to contribute to the development of poor quality elements [34]. Smoothing

techniques have been implemented to improve upon the mesh quality and have had some success with resolving irregular nodes [6].

There are two main indirect approaches seen in literature. They are the H-Morph [33] method and a tetrahedra dicing method known as tetrahedra to hexahedra conversion (TTH) [5][6]. A description of each method follows.

H-Morph

The H-Morph algorithm, presented by Owen et al. [33], generates a hexahedral-dominant mesh. H-Morph builds on the Q-Morph algorithm presented by Owen et al. [34], where a triangular mesh is converted into a quadrilateral mesh.

This method starts with an initial tetrahedral mesh and systematically combines neighbouring tetrahedra into hexahedra. It follows an advancing front approach, where the initial front begins at the boundary of a volume and advances towards the interior. This process continues until the entire tetrahedral mesh has been converted or until valid, good quality hexahedral elements can no longer be created. Any remaining tetrahedral elements tend to be buried within the interior of the volume. The interior region of a volume is generally considered to be less critical for analysis [33]. Obviously the associated solver must be able to handle both tetrahedral and hexahedral elements.

The combining of tetrahedral elements to make a hexahedron usually requires five or more tetrahedrons. Figure 2.13 gives an example of how tetrahedral elements combine to make a hexahedron.

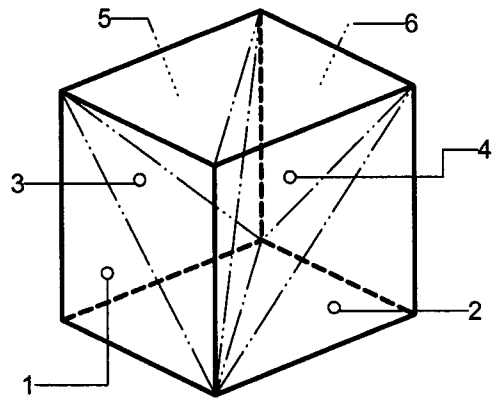


Figure 2.13: Construction of a hexahedron from 6 tetrahedra [33].

The H-Morph method is still relatively new and has not been tested over a wide range of geometries. It has proven itself capable of creating and maintaining valid hexahedral dominant meshes throughout its process, but further testing of this scheme is still required [33].

Tetrahedra to Hexahedra Conversion (TTH)

TTH conversion [6], also known as dicing [5], is another method of indirectly generating hexahedra elements. This technique sub-divides a tetrahedra element into 4 hexahedra elements. This technique works by creating ten new nodes and positioning them at the centre of each edge and face of a tetrahedral element. A last node is inserted within the volume of the tetrahedron itself. Edges are connected between these newly inserted nodes to form the 4 new hexahedra as seen in Figure 2.14.

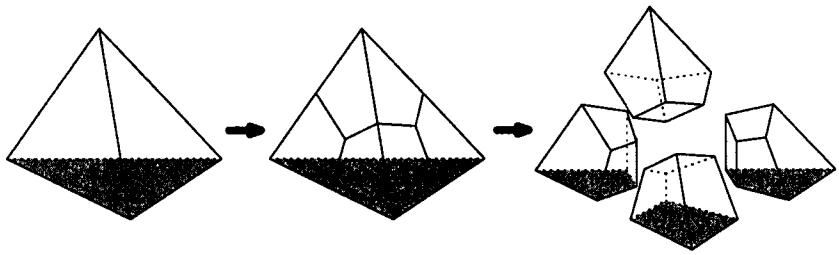


Figure 2.14: TTH conversion (dicing) of a tetrahedron into four hexahedra [3].

The main benefit of using a TTH scheme is its ability to quickly mesh complex solid volumes with an all-hexahedra mesh. It does require the object to be meshed into tetrahedra elements prior to use, however, freemeshing tools are common in most CAD/FEA software packages [1].

Although starting from a tetrahedral mesh allows a TTH algorithm to avoid using computationally expensive subdivision routines, it also means the hexahedral mesh cannot be, initially, of better topological quality than the original mesh. Essentially, this means that the hexahedral mesh cannot conform better to the geometry than that of the original tetrahedral mesh.

The main disadvantage of using TTH schemes is the poor quality hexahedral elements produced. Because the hexahedra are made directly from tetrahedral elements, their shapes tend to be distorted. This distortion is compounded when poor quality tetrahedral elements are contained within the mesh. To compensate for this, smoothing algorithms are used to improve the overall mesh quality [6].

A secondary drawback to this method results from the subdivision of elements. By converting each tetrahedron within the mesh into 4 hexahedrons, the mesh size is effectively increased by a factor of four. This refinement has the possibility of making any well conforming mesh much more refined than needed for an analysis. Overly

refined meshes are detrimental because they greatly increase the computation resources and processing time required for analyzing FEA problems; especially for nonlinear cases. It is for this reason that an algorithm for coarsening TTH converted meshes is believed to be beneficial.

2.3 Mesh Management

Once a volume has been discretized, it is common for there to be poor quality elements within the mesh. Poor quality elements are known to affect the accuracy of FEA solutions and should be eliminated whenever possible [35]. For this reason, a number of techniques have been developed to improve the quality of a mesh. Most techniques involve using some form of metric for measuring the distortion of an element, and then attempting to improve distorted elements through smoothing or cleanup operations [3].

For good quality meshes, there are additional methods used for further improving an analysis. These techniques involve either refining element sizes or increasing the shape function order by adding additional nodes to elements.

A description of mesh quality metrics, mesh improvement and refinement techniques are discussed below.

2.3.1 Assessing Mesh Quality

Although element distortion cannot be directly attributed to error in every analysis, it has been shown that a better quality mesh will give superior results overall [35]. The distortion of an element occurs when the shape of an element in the physical (x, y, z) domain deviates from the orthogonal isoparametric element in computational (r,

s, t) space [35]. To quantify distortion, a number of metric functions have been developed. Table 2.1 lists a few metrics used to measure element distortion and their full and acceptable ranges. Knupp [36][37] has summarized these and other quality metrics in an effort to compare which are the most promising for use on difficult problems. He found that none of the metrics excelled when compared to the others on a range of 3-D volume meshes.

Table 2.1: Metrics for hexahedral elements with functional ranges [24][35][36][37].

Function Name	Dimension	Full Range	Acceptable Range
Aspect Ratio	L^0	1 to infinity ($+\infty$)	1 to 4
Skew Angle	L^0 (degrees)	0° to 180°	45° to 135°
Taper Ratio	L^0	0 to infinity	0 to 0.4
Det. Jacobian	L^3 (volume)	Infinity ($-\infty$) to infinity ($+\infty$)	Positive (+)
Oddy Metric	L^4	0 to infinity ($+\infty$)	≤ 10
Stretch	L^0	0 to 1	0.25 to 1
Diagonal Ratio	L^0	0 to 1	0.65 to 1
Distortion Parameter	L^0	-1 to 1	>0
Condition Number	L^0	1 to infinity ($+\infty$)	1 to 8

The TTH dicing algorithm [6] employs the aspect ratio, skew angle and determinant of the Jacobian, as metrics to measure mesh quality. These metrics were believed to give a good indication of mesh quality. The determinant of the Jacobian (det *J*) is a good indicator for flagging when an element becomes inverted. The Jacobian is a matrix of partial derivative equations of the physical (*x, y, z*) domain with respect to computational (*r, s, t*) space. Equation 2-1 shows one form of the 3-D Jacobian matrix

$$[J] = \begin{bmatrix} \partial x/\partial r & \partial x/\partial s & \partial x/\partial t \\ \partial y/\partial r & \partial y/\partial s & \partial y/\partial t \\ \partial z/\partial r & \partial z/\partial s & \partial z/\partial t \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} & J_{13} \\ J_{21} & J_{22} & J_{23} \\ J_{31} & J_{32} & J_{33} \end{bmatrix} \quad (2.3)^4$$

Its determinant is evaluated at the centroid and at all eight Gauss points of an element. For an 8-node hexahedron, Gauss points are typically located at coordinates:

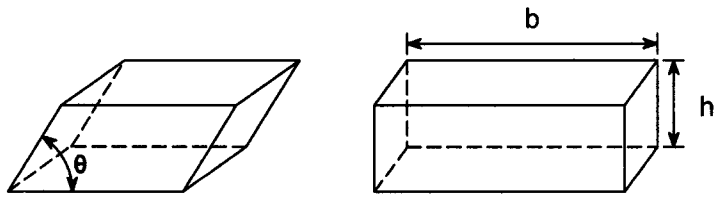
$(r = \pm 0.577, s = \pm 0.577, t = \pm 0.577)$.

The equation for finding the determinant of the Jacobian is as follows:

$$\begin{aligned} \det J = \{ & J_{11} \times [(J_{22} \times J_{33}) - (J_{23} \times J_{32})] + J_{12} \times [(J_{23} \times J_{31}) - (J_{21} \times J_{33})] \\ & + J_{13} \times [(J_{21} \times J_{32}) - (J_{22} \times J_{31})]\} \end{aligned} \quad (2.4)$$

A negative det J means that the element has a negative volume and is considered 'inverted'. A mesh with inverted elements is invalid and cannot be used to analyze geometry. Any inverted elements must be straightened out by using an untangling algorithm [38].

The skew angles and aspect ratios can be found through decomposing the Jacobian using Kerlick and Klopfer [39]. The skew angles are a measure of the rotation between the (r, s, t) natural space coordinates with respect to the (x, y, z) physical space coordinates. Figure 2.15 (a) gives an illustration of a skew angle.



a) Skew Angle (θ) b) Aspect Ratio (b/h)

Figure 2.15: Examples of hexahedral distortion.

[4] Alternative forms of the Jacobian exist. In some references, the transpose of equation (2.3) is used as the Jacobian. Either form may be used, as long as it is used consistently.

Both skew angles and aspect ratios are found by decomposing the Jacobian through its metric tensor (g). The metric tensor, equation (2.5), is the square of the Jacobian matrix, and is considered an:

"intrinsic property of the computational space, independent of its embedding" [39].

$$g = J^T J = \begin{bmatrix} \partial x/\partial r & \partial y/\partial r & \partial z/\partial r \\ \partial x/\partial s & \partial y/\partial s & \partial z/\partial s \\ \partial x/\partial t & \partial y/\partial t & \partial z/\partial t \end{bmatrix} \times \begin{bmatrix} \partial x/\partial r & \partial x/\partial s & \partial x/\partial t \\ \partial y/\partial r & \partial y/\partial s & \partial y/\partial t \\ \partial z/\partial r & \partial z/\partial s & \partial z/\partial t \end{bmatrix} = \begin{bmatrix} g_{11} & g_{12} & g_{13} \\ g_{21} & g_{22} & g_{23} \\ g_{31} & g_{32} & g_{33} \end{bmatrix} \quad (2.5)$$

where,

$$\begin{aligned} g_{11} &= (\partial x/\partial r)^2 + (\partial y/\partial r)^2 + (\partial z/\partial r)^2 \\ g_{22} &= (\partial x/\partial s)^2 + (\partial y/\partial s)^2 + (\partial z/\partial s)^2 \\ g_{33} &= (\partial x/\partial t)^2 + (\partial y/\partial t)^2 + (\partial z/\partial t)^2 \\ g_{21} &= g_{12} = (\partial x/\partial s)(\partial x/\partial r) + (\partial y/\partial s)(\partial y/\partial r) + (\partial z/\partial s)(\partial z/\partial r) \\ g_{31} &= g_{13} = (\partial x/\partial t)(\partial x/\partial r) + (\partial y/\partial t)(\partial y/\partial r) + (\partial z/\partial t)(\partial z/\partial r) \\ g_{32} &= g_{23} = (\partial x/\partial t)(\partial x/\partial s) + (\partial y/\partial t)(\partial y/\partial s) + (\partial z/\partial t)(\partial z/\partial s) \end{aligned} \quad (2.6)$$

The Jacobian, which relates physical and computational space, provides 3 skew angles as defined by the physical coordinates and the computational axes. The skew angles are found by using equation 2.7 [39]:

$$\cos\theta_{ij} = \frac{g_{ij}}{\sqrt{g_{ii} \times g_{jj}}} \quad \text{where, } i, j = 1, 2, 3 \text{ and } i \neq j \quad (2.7)$$

The principle aspect ratios, also provided through the metric tensor of the Jacobian, define the ratio of the tangent vector magnitudes [39]. Figure 2.15 (b) gives an illustration of the aspect ratio.

$$AspectRatio(AR) = \sqrt{\frac{g_{ii}}{g_{jj}}} \quad \text{where, } i,j = 1,2,3 \text{ and } i \neq j \tag{2.8}$$

2.3.2 Smoothing

Smoothing techniques attempt to improve the shape of an element by iteratively adjusting nodal coordinates until they meet set criteria. The criteria are determined by using one of the metrics mentioned above. These procedures keep the element connectivity intact during node movement as compared to cleanup algorithms, which change element connectivity. Smoothing algorithms are generally classified as one of four categories: 1) Averaging methods, 2) Optimization-based methods, 3) Physically-based methods and 4) Midedge node placement methods.

Averaging methods reposition corner nodes based on a weighted average of the geometric properties of neighbouring nodes and elements. The most common averaging technique is Laplacian smoothing [40]. Laplacian smoothing is easily implemented and generally works well for elements in convex regions. As these techniques have the occasional drawback of creating inverted elements, constraints are imposed in the algorithms to limit node movement [3][40].

Optimization-based methods reposition corner nodes based on a distortion metric. These methods measure the quality of elements surrounding a node and compute the local gradient of the quality of the elements. The node is then repositioned in the direction of the increasing gradient until an optimum is achieved. Optimization-based algorithms generally provide superior improvements to mesh quality, but they also are computationally expensive [3]. Canann [40] and Freitag [41] have presented algorithms

in which a combined Laplacian/optimization base approach is used to reduce overall computational time.

Genetic Algorithms (GA) are a recent addition to optimization-based mesh smoothing algorithms. They operate on populations of solutions and progressively modify their chromosomes (binary coding of solutions), in the hope of improving their fitness (quality metric). Much like natural evolution, these algorithms try to iteratively reproduce a population through mixing the best solution chromosome sets together to see if their children are of better fitness (quality) [42][43][44][45]. Holder [46] has presented a method for using GAs to optimize a mesh using the Oddy [35] metric as a fitness function. Genetic algorithms are found to produce good solutions to problems, but do not necessarily find the best solution. They also are very computationally expensive, more so than other gradient based optimization routines [43].

Physically-based methods reposition corner nodes based on simulated physical attraction/repulsion forces between nodes. These approaches can produce different anisotropic element shapes and sizes depending on the magnitude and direction of the simulated forces.

Midedge node placement methods are designed for quadratic elements. They smooth quadratic elements by shifting midedge nodes within a defined admissible space [47]. The midedge node admissible space (MAS) is calculated from a metric that is able to detect poor and good quality elements, in spite of the fact that geometric distortion is not necessarily the best indicator of quality for quadratic elements.

2.3.3 Cleanup

Unlike smoothing techniques, cleanup algorithms improve the quality of a mesh through local changes to element connectivity. These algorithms use techniques such as node insertion, edge/face swapping, node removal or element deletion to improve criteria such as element shape or topology [40]. Element shape improvement deals with changing the connectivity of elements to increase their quality. Topology improvement is the attempt to optimize the valence of irregular nodes within the mesh [48]. The valence is the number of edges that a node shares with other nodes. For hexahedral meshes, internal irregular nodes have a valence of more or less than 6. To improve a nodes valence, local transformations are performed to bring the valence as close to 6 as possible.

2.3.4 Refinement

The term refinement is being used here in a broad sense to cover both refining and coarsening operations on a mesh. Refinement operations are meant to improve the meshing conditions for capturing critical information while maintaining reasonable processing costs. These algorithms can be used to uniformly refine the complete mesh or act upon select regions. This is known as mesh grading or biasing. Mesh grading is often used for problems with localized phenomena, such as point loads, point supports, crack tips, moving loads or heat sources [15]. Refinement schemes fall under a number of strategies, examples of which are h-refinement, p-refinement, hp-refinement and r-refinement.

H-refinement can be described as any operation performed on a mesh that reduces an elements size, and thus increases the mesh density. This is typically done by sub-

dividing elements into smaller sections. The opposite of h-refinement is h-coarsening, where elements are merged together into larger elements.

P-refinement increases the polynomial order of the basis functions. The element size is kept fixed, but extra nodes or higher order modes are added to the element. The solution convergence rate for p-refined meshes is better than h-refined meshes for smooth functions, however p-refinement becomes less effective then h-refinement at converging around singularities such as corners or point loads [49].

The hp-method combines both h- and p- refinement schemes. These hybrid methods are used to overcome the difficulty p-refinement has with converging around singularities [49].

R-refinement adjusts the node locations within natural space, essentially deforming the master element.

H-refinement techniques are the most commonly used methods, which can be seen through the number of commercial FEA packages that employ such methods [1]. The focus of this thesis falls in the area of h-coarsening; therefore more detail on this subject is presented below.

2.3.4.1 H-Coarsening

As mentioned above, h-refinement/coarsening techniques have been developed to manage the density of a mesh with the aim of finding an acceptably accurate solution with minimum computational cost. While refinement algorithms are used to improve computational accuracy, coarsening routines are meant to improve computational cost by decreasing the mesh density in localized regions or globally throughout a domain.

H-refinement/coarsening schemes have been implemented in adaptive remeshing routines and as part of pre-processor mesh generation applications. Adaptive routines refine and coarsen a mesh during the solution of a problem, in an attempt to decrease convergence time. These algorithms use error norm calculations to determine areas of high and low error gradients. The mesh is then refined in areas of high error gradients and coarsened in regions of low error gradients [50][51].

Refinement/coarsening schemes employed within the pre-processor stage are typically manual algorithms. An analyst selects regions of a mesh where to apply these schemes. Coarsening algorithms are not as prevalent as refinement methods since most analysts will start with a coarse mesh and refine it until they reach an acceptably accurate solution. Few hexahedral h-coarsening algorithms are seen in literature since most hexahedra research has gone towards developing a robust and unattended autonomous hexahedral mesh generator. The h-coarsening algorithm presented within this thesis, attempts to meet the need for a tool capable of selectively coarsening irregular, distorted hexahedral meshes.

The Cubit software [24], which is a state-of-the-art meshing research application, provides limited capabilities for coarsening hexahedral meshes. It relies upon a hex sheet extraction process, wherein a sheet of elements is removed and their neighbours are expanded to take their place. This technique works well for well-shaped, regular meshes where a layer of elements can be established for removal. It is not so easily implemented over irregular meshes generated by applications such as TTH. Nor is it suitable for localized mesh grading.

McDill et al. [52] presents an arbitrary coarsening algorithm for hexahedral meshes. It uses the 8- to 26-node isoparametric linear brick [15]. Although the coarsening routine proposed by McDill was designed as part of an adaptive remeshing tool, it can be applied to coarsening meshes at the pre-processor stage. This arbitrary coarsening algorithm allows 2, 4 or 8 element siblings to amalgamate into a single parent element. The connectivity to all surrounding elements is kept by making neighbouring element's vertex nodes into edge or face nodes on the new parent element. Figure 2.16 show a mesh arbitrarily coarsened in localized regions.

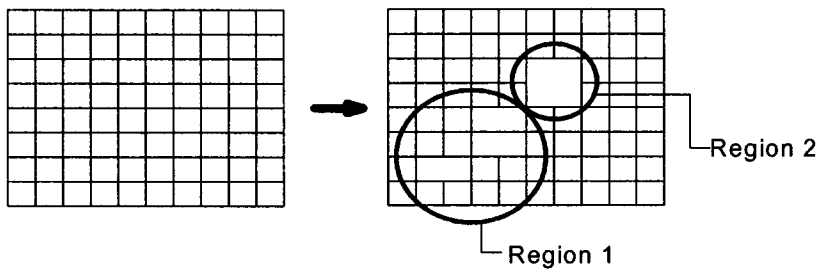


Figure 2.16: Arbitrary h-coarsening of a 2-D quadrilateral mesh

In Region 1 of Figure 2.16, two elements have been merged into a larger parent. Region 2 shows 4 elements merged into a parent element. Any neighbouring vertex nodes that are not vertex nodes for the new element become midedge or midface (3-D) nodes on the new element. Alternatively, external constraint equations can be applied.

As seen in Figure 2.16, this technique is able to efficiently grade a mesh with little to no deformation of the surrounding elements. Although this technique was designed for regular, minimally distorted brick meshes; there is promise of extending it to work with irregular, distorted meshes.

Chapter 3: TTH-GENERATED MESHES

Before introducing the coarsening approach taken, it is useful to review the principal steps the TTH method [6][53] uses to generate a hexahedral mesh. Following this, the challenges with coarsening TTH-generated hexahedral meshes are explored.

3.1 Review of TTH Software

The tetrahedra-to-hexahedra (TTH) conversion software [6] was developed to investigate the dicing technique as an alternative means of generating an all-hexahedral mesh. This technique was originally thought to produce poor quality hexahedra, and hence, had been thought to be unreliable for FEA analysis. This type of thinking has recently changed with studies showing that this is not necessarily true for particular analysis types [5].

The TTH software was programmed in C (language) and consists of four main modules: 1) data gathering, 2) dicing, 3) quality assessment and 4) smoothing. The data-gathering module extracts the original tetrahedral mesh data. This module is unrelated to the quality of the mesh conversion and will not be discussed further. The remaining three modules directly influence the outcome of the mesh. These modules are summarized below.

3.1.1 Dicing

The dicing module systematically splits each linear tetrahedral element within a mesh into 4 linear hexahedral elements. The method used by TTH for dicing a tetrahedron is described below:

1. Place one node at each midedge of the tetrahedron (6 nodes total).
2. Place one node at each midface of the tetrahedron (4 nodes total)
3. Place a single node within the volume (interior or midbody) of the tetrahedron.[5]
4. Connect the midface nodes to their three adjacent midedge nodes and the midbody node.
5. Use the links to divide the tetrahedron and establish 4 hexahedral elements.

Figure 3.1 (a) illustrates the new node placement and the links between the midedge, midface and midbody nodes. Figure 3.1 (b) depicts the four new hexahedral elements.

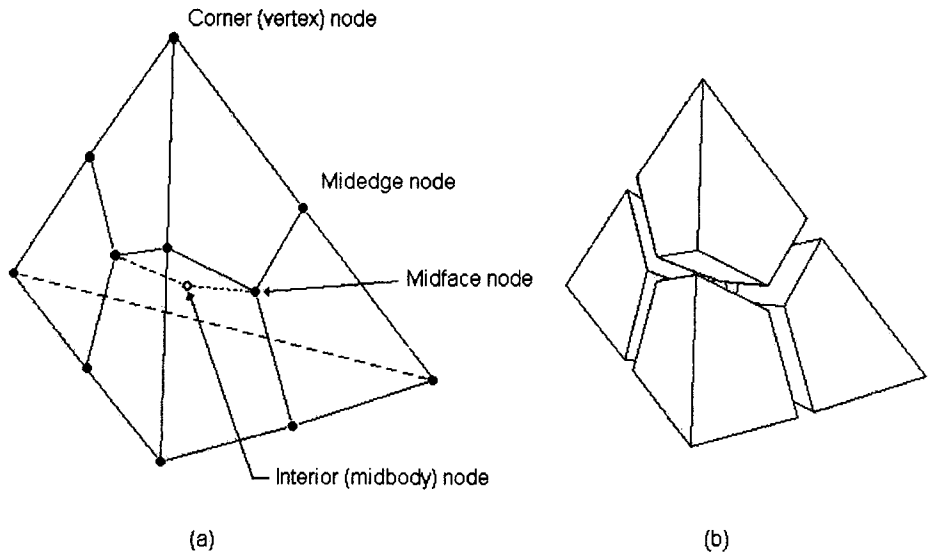


Figure 3.1: Tetrahedra-to-hexahedra dicing process. (a) Midedge/midface/midbody node placement (b) Resulting hexahedra

[5] The TTH software places the interior node at the center of volume and then adjusts its position based on the aspect ratio of the tetrahedra.

3.1.2 Quality Assessment

As described within the previous chapter, TTH employs three quality metrics (checks) to measure the quality of hexahedra. It utilizes skew angle and aspect ratio checks to determine when to perform smoothing operations on an element and uses an element verification check to prevent elements from becoming inverted.

Ideal hexahedral elements have skew angles of 90° and aspect ratios of 1. As seen in Figure 3.1 (b), TTH does not create ideal hexahedral elements. As the shape of the hexahedra are governed by the initial shape of their tetrahedron, all TTH-generated hexahedra will be non-orthogonal. Also, if a tetrahedral element is of poor quality, the hexahedra created from it will be of poor quality. Clearly, better quality hexahedra will typically come from good quality tetrahedral meshes.

To determine when a hexahedral element needs smoothing performed, TTH uses two checks. It checks the skew angles and aspect ratios of each element. If either metric is out of tolerance, then smoothing is performed. TTH utilizes acceptable hexahedron tolerances of less than 2.5 for aspect ratios and between 45° to 135° for skew angles. The aspect ratio tolerance was chosen to be slightly more conservative than the tolerances outlined by Oddy [35].

For elements undergoing smoothing operations, there is the possibility of an element becoming inverted. To ensure this does not happen, TTH employs an element verification check. The element verification check calculates the determinant of the Jacobian, equation (2.4), of an element at its centroid and at each of its eight Gauss points. If the determinant of the Jacobian is less than or equal to zero at any of these locations, then the element is considered to be inverted. Any node adjustments that have

led to the inversion of an element are removed, returning each node to its previous position.

3.1.3 Smoothing

TTH incorporates three different routines to smooth poor quality hexahedra. Each routine performs a specific operation to improve either the skew angle or the aspect ratio of an element. The routines included within TTH's smoothing operations are: Skew Rotate, AR Fix and Isoparametric Smoothing. The smoothing operations were limited to working with interior nodes. Exterior (surface) nodes were restricted from undergoing smoothing since their movement would distort the original shape of the geometry.

The first of these routines, Skew Rotate, improves the skew angles of all elements outside of the 45° to 135° angle tolerance. Skew Rotate is a geometry-based smoothing routine designed to move opposite faces of a hexahedra along parallel planes in an effort to bring the worst skew angles within tolerance. Geometry-based smoothing techniques usually fall under the category of untangling algorithms. Untangling algorithms attempt to eliminate inverted elements. In this case, the geometry-based routine is used to reduce element distortion instead of element inversion. Figure 3.2 illustrates the motion of opposite facial nodes of an element, along parallel planes to reduce poor skew angle measurements.

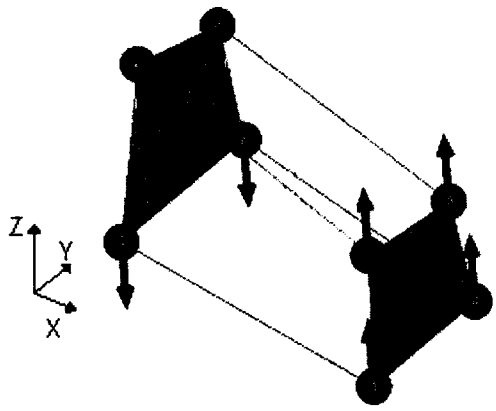


Figure 3.2: Skew rotation of a hexahedral element [6].

The second routine, AR Fix, improves the aspect ratio of all elements with ratios greater than 2.5. AR Fix is another geometry-based smoothing routine, designed to work similarly to Skew Rotate. Instead of moving opposite faces along parallel planes, AR Fix reduces or extends the distance between opposite faces depending on their length-to-width ratios (aspect ratios). Figure 3.3 depicts the motion of the nodes on opposite faces to reduce excessive aspect ratios.

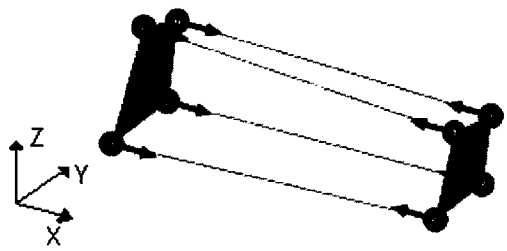


Figure 3.3: AR fix of a hexahedral element. [6]

The final routine, Isoparametric Smoothing, is a hybrid Isoparametric-Laplacian method which operates on the principle of average node position. This method was first proposed by Herrmann [54] for 2-D geometries and was extended to work in 3-D within

TTH [6]. TTH used the following 3-D isoparametric equation, derived from Herrmann's original equation, to direct isoparametric smoothing:

$$x' = \frac{1}{3*NE} * \left(\sum_{1}^{NE} w*(x_A + x_B + x_C) + 3*(1-w)x_D \right) \tag{3.1}$$

where, x' is the obtained coordinate for the intended node after smoothing,

x_A, x_B, x_C are the adjacent node coordinates to the intended node for smoothing,

NE is the number of neighbouring elements connected to the intended node,

x_D represents the element node that is in the opposite corner to the intended node,

w represents a weight function with a practical range of 0 to 1.

The weight function (w) in (3.1) controls the transition of the equation from Laplacian (w = 0) to isoparametric (w = 1). The TTH software uses a weight function value of w = 0.7. Although this value was shown to provide a stable balance of performance versus time spent smoothing, it would be beneficial to perform a study to optimize this value.

Unlike the previous two smoothing routines, this method smoothes the mesh on a node-by-node basis [6]. The Isoparametric routine was implemented to complement the previous two geometry-based routines. This routine loops through each node within an element, shifting its position so that the shape of the element becomes more orthogonal.

3.2 Coarsening Challenges

The topography and connectivity of TTH-generated meshes leads to some interesting challenges for coarsening algorithms. Since TTH-generated meshes originate from tetrahedral meshes, the majority of the nodes within a TTH-generated mesh are

irregular; i.e., they have more or less than eight neighbouring elements (for interior nodes) or four neighbouring elements (for surface nodes). For the purpose of this thesis, all elements connected to a node are considered to be neighbours. Figure 3.4 illustrates a TTH-generated mesh.

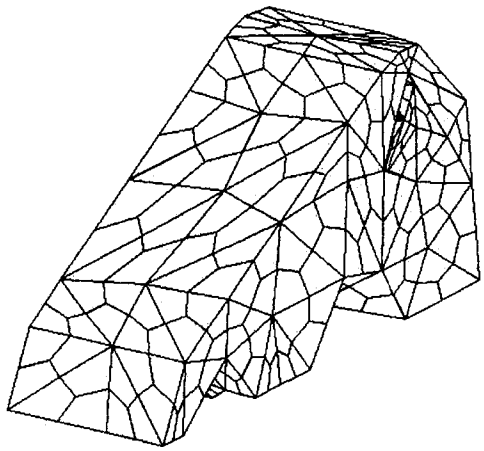


Figure 3.4: A TTH-generated mesh. [6]

It is apparent from this illustration that there are certain nodes on the surface that have five or more neighbouring surface elements. However, what is not apparent is that these same nodes have a number of interior elements neighbouring them from below the surface. For some TTH-generated meshes, there may be upwards of 40 elements neighbouring a node. This is due to the algorithm used by the mesh generation software when creating the original tetrahedral mesh. Irregular nodes and their respective number of neighbouring elements are a concern because they indicate the general distortion of a hexahedral mesh. Although TTH-generated meshes undergo a smoothing operation, the number of irregular nodes limits the improvements these routines can produce.

The challenge TTH imposes on coarsening algorithms is twofold. First, the algorithm must take into account element distortion and what effects coarsening will have

on the surrounding region. With well-shaped hexahedral meshes, h-coarsening algorithms merge elements into a larger parent element, but will generally not change the shape of the elements or any of their neighbours. This can be seen in Figure 2.16. For meshes with irregular nodes and highly distorted elements, this is not the case. Joining distorted elements will require a change in local geometry of the surrounding elements. Therefore, the coarsening algorithm must perform surface topology and binary-grading checks for limiting the changes that might adversely affect the surrounding elements and any surfaces they may contact. These checks will be discussed in later chapters.

Secondly, the dicing method employed by the TTH algorithm creates a unique challenge for coarsening algorithms, specifically for the 2-element coarsening scheme presented in this thesis. When two hexahedral elements from the same parent tetrahedra are merged into a single element, they will deform the remaining two elements from the same parent tetrahedra, into wedge shaped (six-sided) elements. Figure 3.5 demonstrates this occurrence. The top surface of the parent tetrahedral element in figure 3.5 is enhanced to show node positions. In Figure 3.5 (a), a parent tetrahedra is shown divided into 4 hexahedral elements. When two of the hexahedral elements merge, Figure 3.5 (b), the new element straightens its edges by moving its midedge nodes[6]. Since these midedge nodes are corner nodes for neighbouring elements, this effectively moves two adjacent faces of an element into the same plane. As seen in Figure 3.5 (c), these two surrounding elements become wedge-shaped.

[6] When two sibling elements merger into a parent element, their four shared corner nodes become midedge nodes on the new parent element. The parent element is an 8- to 26-node isoparametric hexahedral element presented by McDill [15].

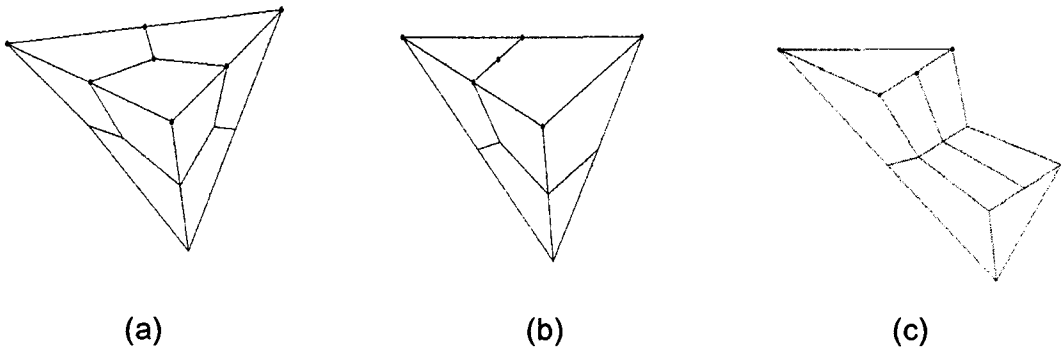


Figure 3.5: Wedge-shaped element due to coarsening.

Wedge-shaped elements are undesirable because they are severely distorted hexahedral elements. They differ from other distorted elements because their aspect ratios and skew angles are normally found to be within the tolerances established earlier. For an algorithm to coarsen a TTH-generated mesh effectively, it must be able to detect and avoid creating these types of elements.

In summary, for coarsening TTH-generated meshes, the following issues must be dealt with:

1. Detect and refuse coarsening that causes changes to the surface topology of the geometry or causes element connectivity to be lost.
2. Detect and avoid creating severely distorted elements such as inverted or wedge-shaped elements.

Chapter 4: COARSENING METHODOLOGY

Most automated, hexahedral h-coarsening routines that exist in literature require a mesh with well-shaped elements. Meshes with poor quality elements are more difficult to coarsen, due to the problems associated with merging non-orthogonal elements. As seen in the previous chapter, TTH-generated meshes consist of non-orthogonal, distorted elements. This chapter presents detailed information on the algorithm developed to coarsen a TTH-generated hexahedral mesh.

The coarsening algorithm has been designed to utilize the mesh format employed by the specialty FEA code TIAMAT. The TIAMAT FEA software supported mesh grading through the use of 8- to 26-node linear isoparametric hexahedrons [15].

As the elements generated by coarsening needed to work within TIAMAT, emphasis was placed on developing a methodology that would interface well with existing approaches and software.

4.1 Arbitrary Coarsening Background

The arbitrary coarsening algorithm presented by McDill et al. [52], combines 2, 4, or 8 sibling elements into a single parent element. This scheme was developed to avoid the hierarchical parent-child relationship (family tree structures), common to many mesh-management algorithms [55][56] [57][58].

Hierarchical family tree schemes require an initial root (parent) mesh from which the refined (graded) elements have been tracked through their parent-child divisions. Hierarchical coarsening uses the family tree to reproduce a parent element from its children. Since the mesh generated by TTH does not contain a hexahedral family tree

data structure, a parent-child scheme would not to be able to coarsen any elements. Therefore, an arbitrary scheme is necessary for coarsening TTH-generated meshes.

Arbitrary schemes use adjacency relationships to match neighbouring elements for coarsening. In the case of 8-element coarsening, each element must share the same node. Coarsening 4-elements requires a shared edge and 2-element coarsening requires a shared face. The resulting parent element becomes an 8- to 26-node linear isoparametric hexahedron [15] or nonconforming hexahedron [16] to permit mesh grading without the use of external constraint equations.

McDill et al. [52] outlined a set of rules that must be met before elements may undergo coarsening. These rules can be modified to work in the case of coarsening TTH-generated meshes with adaptive mesh-management constraints removed. The modified rules are listed below.

Rule C1: 2, 4 or 8 sibling elements may be coarsened into a single parent element.

Rule C2: Siblings may not coarsen into a severely distorted parent.

Rule C3: The creation of a new parent element may not cause severe distortion to neighbouring elements.

Rule C4: The mesh grading may not exceed a ratio of 2-to-1 to preserve element connectivity and avoid rapid transitions from fine to coarse regions.

Rule C5: Coarsening that alters the exterior geometry of the object is not allowed.

Rule C6: Redundant nodes; i.e. those not required to represent topology or geometry, are removed. Nonredundant nodes must be retained.

4.2 Coarsening Outline

The main goal for this coarsening algorithm was to develop an application that is able to coarsen regions within an irregular, non-orthogonal mesh. The following sections

will detail the approach taken to identify and implement solutions to various challenges discovered with coarsening TTH-generated meshes.

A 2-element coarsening routine was selected as the initial starting point. This routine was chosen because it was believed to be easier to implement than 4- or 8-element coarsening. Also, the development of this routine would create a framework from which the more difficult 4- or 8-element coarsening routines could be built.

The method for selective coarsening was determined to be most easily implemented through a graphical user interface (GUI). Therefore, a GUI application named GITTH (Graphical user Interface to TTH), was developed to assemble TTH and all of its peripheral software into a single application. The coarsening functionality was included within GITTH through a mesh-management module. The mesh-management aspect of GITTH was designed to work in conjunction with LS-PrePost[7]. LS-PrePost is a pre-/post-processor used by TTH to visualize the mesh. GITTH is described to a greater extent in Chapter 5 with full software documentation enclosed within Appendix 1: GITTH User Manual.

The mesh-management module was designed to support the input of node numbers by an analyst, as the means for selective coarsening. Node numbers were selected as the best approach for selecting an area within the mesh for coarsening, since they do not require re-ordering after elements are merged. TIAMAT requires the element information to have continuous and sequential numbering, but does not require this of the node numbering. Due to this requirement, element numbers must be re-ordered to

[7] LS-PrePost is a commercial pre/post-processor developed by the Livermore Software Technology Corporation (LSTC) for PC and Linux stations. It has been developed by LSTC to work as a visualization tool for the commercial FEA application, LS-DYNA. LS-PrePost is the upgraded version of TAURUS, which was used previously for post-processing of TIAMAT simulations.

remove invalid element information due to coarsening and to preserve a continuous element numbering system. The alternative method for selecting an area is through the input of an element number. This method is not as stable as node selection, since element numbers must be re-ordered after each coarsening run. Element re-numbering will be described in greater detail in Section 4.9.

Figure 4.1 presents the main steps within the coarsening process. The process starts with the manual input of a node and a coarsening code selection (2, 4 or 8 elements for coarsening). At this stage of development, only the 2-element coarsening algorithm has been completed. Next, the adjacency relationships (element connectivity) and element surface determinations are established. This is followed by edge checks to ensure that the node selected is not situated upon an exterior edge or material border within the geometry. If the node is not sitting upon an edge, the 2-element coarsening module is run. The 2-element module searches for two elements to merge into an acceptable parent. If two elements are found to meet the rules laid out in Section 4.1, they are merged into a single parent element. The second last step searches for any redundant nodes created through the coarsening process, removing them from the mesh. The last step renumbers the element identification numbers, removing any invalid element information.

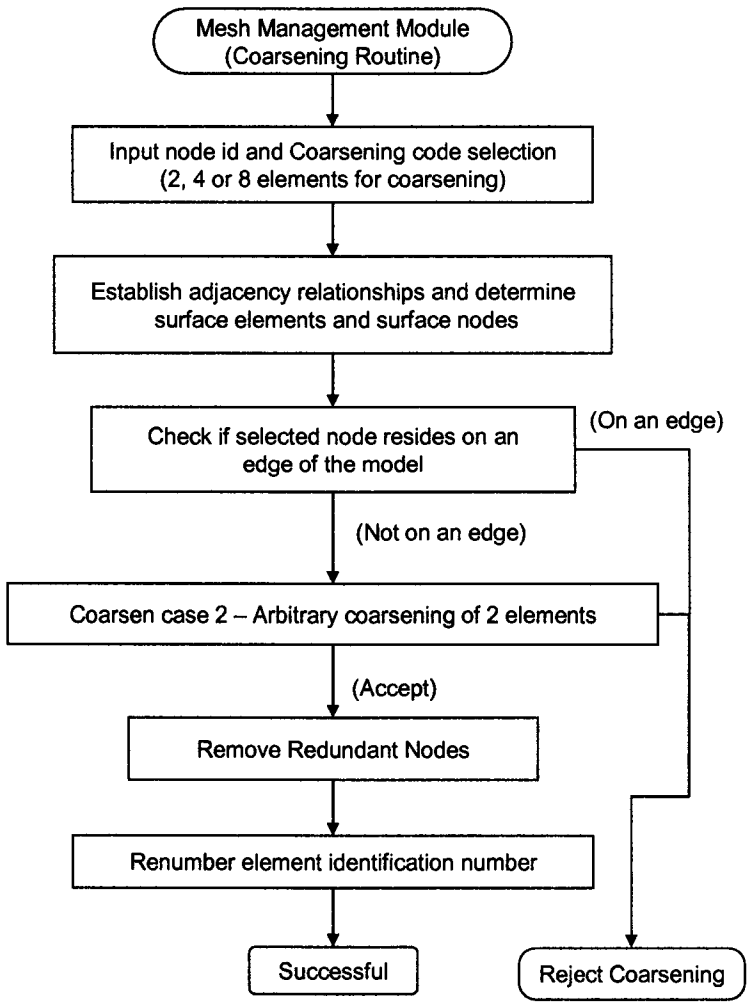


Figure 4.1: Coarsening Process

4.3 Programming Language

The mesh-management (coarsening) module was written using the programming language C and tested using a Linux (Red Hat 9.0) system complier. C was chosen for a number of reasons.

First, the mesh-management routine and the GITTH interface both were required to work on Linux operating systems. Linux platforms are created with C and have their own native C compiler. These platforms also contain many open source C libraries

which were useful for supporting the development of a GUI application and any software module running off the GUI.

Second, TTH was written in C. Keeping a constant programming language across an application made it much easier for software modules to interact. It also ensured the compatibility of the mesh data structure and allowed sharing of functions.

Last, C provided the ability to handle large multi-dimensional arrays and supported dynamic memory allocation. The multi-dimensional array functionality provided by C offered an efficient means of storing mesh data. Since the coarsening algorithm must handle reasonably large mesh sizes, multi-dimensional arrays were effective for managing element and nodal information. This is discussed in more detail in Section 4.4. Dynamic memory allocation was very useful for allocating and freeing memory on demand. It was used extensively to ensure the GITTH application had the memory resources it needed.

4.4 Data Structure

The coarsening algorithm was developed utilizing the same read/write mesh data file format employed by TTH. This file format was initially established for the TIAMAT FEA software. The mesh data file format handles an 8- to 26-node element; recording topology, connectivity, material codes and TIAMAT specific analysis codes for thermal-mechanical analysis. The mesh file format is explained in detail within the referenced document [59].

The maximum allowable size for hexahedral meshes was hard coded into TTH and carried over into the coarsening module. After considerable testing, Carmona [6]

limited the variable MAX_NO_EL (the number of elements) to 46,000 and the variable NODMAX (the number of nodes) to 66,000. These limits were based on the maximum mesh sizes undergoing evaluation and the computer resources present at that time. The limits shown here were found to be acceptable for testing the coarsening software over a wide range of mesh cases. If required, these limits can easily be altered to allow for larger meshes or reduce memory resource requirements.

The mesh data, which includes element and node information, is stored in two separate multi-dimensional global space arrays. These arrays were placed in global space so that they would remain in memory throughout the duration of the applications runtime and would be accessible to all sub-routines. The element data array is called elementData[MAX_NO_EL][28] and the nodal information array is called nodeData[NODMAX][4]. The elementData array holds up to MAX_NO_EL (46,000) elements and was sized to include an element identification number, 26 node numbers and the material code for the element. As mentioned earlier, McDill's 8- to 26-node hexahedron [15] has been used to enable mesh grading during coarsening. Position [0] of the array, contains the element identification number. The following positions [1 to 26] contain the nodal makeup (8 vertex nodes, 16 optional edge nodes and 6 optional face nodes) of the elements, with the last position [27] containing the material type for the element. This data structure was utilized for its efficient and convenient manner for accessing data. The nodeData array holds up to NODMAX (66,000) nodes and was sized to store a node identification number and three global (x, y, z) coordinates. Position [0] contains the node identification number and positions [1 to 3] contain the *x, y, z* coordinates respectively.

4.5 Adjacency Relationships and Surface Determination

Adjacency relationships were established early in the coarsening scheme. These relationships were used to quickly identify element connectivity between neighbouring elements. The element connectivity was stored within a multi-dimensional array named NAYBORS[NCORN][NODMAX]. This static array was used within TTH and has been incorporated into the coarsening software to reduce duplicating memory demands.

The NAYBORS array stores the element connectivity in the form of node ownership over neighbouring elements. Therefore, element connectivity can be extracted from nodal information; i.e. each column within the NAYBORS array represents a node and contains the identification numbers of all elements neighbouring that node. Carmona [6] set the sizing for the NAYBORS array at NCORN (105) rows with NODMAX (66,000) columns.

Surface element and node determination was also established early in the coarsening process. The array Lsurface[MAX_NO_EL][8] was used to store boundary element information. This information was necessary for limiting the coarsening of elements on the geometry surface. A secondary dynamic array, called surface_nodes, was used to list all nodes lying upon the surface of the geometry. This array was used in conjunction with the Lsurface array to limit any coarsening that would deform the surface of the geometry.

4.6 Edge Check

It was understood early during development that any coarsening of elements on the surface could significantly distort the geometry of the exterior. This distortion

occurred specifically when elements were coarsened over an edge or over highly curved surfaces. Planar surfaces will not be deformed by coarsening. Therefore, to ensure that coarsen conforms to rule C5, an edge check step was implemented.

The edge check subroutine defines an edge to be any location where two geometric surfaces meet or when surface normal vectors change by more than 20 degrees between neighbouring surface elements. Such changes in surface normal vectors are seen on curved or rounded surfaces. The edge check was applied directly to the node entered by the user. If the node was found to reside on an edge, then all coarsening around the node was rejected.

To identify when a node resides on an edge, the edge check subroutine performs three edge checks. If any one of the following checks performed is found to be true, then the node is on an edge.

(1) If the node neighbours two or fewer elements.

(2) If the node resides on an edge of an element where two free faces meet. Free faces are any faces of an element that do not share all four vertex (corner) nodes with another element. Element free faces are what make up the exterior boundary of a mesh.

(3) If the direction of unit normal vectors for element free faces changes by more than 20 degrees between neighbouring elements.

The adjacency relationships established earlier were used to quickly find all neighbouring elements. If there are two or fewer neighbouring elements, Check (1) rejects the node for coarsening.

Check (2) determines if any of the neighbouring elements have free faces. These free faces make up an exterior mesh surface. The Lsurface array is utilized to determine if any of the neighbouring elements contain two or more free faces. If this is true, and if

the node resides on the edge of an element where two of the free faces meet, then coarsening around the node is rejected.

Check (3) calculates the normal unit vector for each of the neighbouring element free faces. The normal vector is found through the cross product of the two vectors V13 and V24. See Figure 4.2 for an illustration of this method. The method for calculating and comparing unit normal vectors for element faces can be seen in greater detail within Appendix 2.

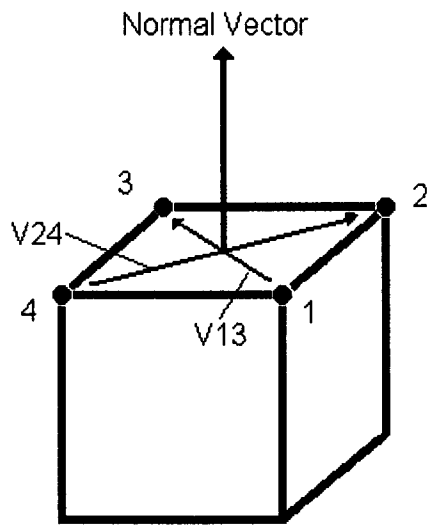


Figure 4.2: Element face normal vector calculation.

If unit normal vectors deviate by more than 20 degrees between neighbouring surface elements, then coarsening around the node is rejected. The 20 degree tolerance for surface variation was found to allow coarsening of relatively smooth surfaces but stopped coarsening of curved surfaces. This tolerance value was taken from the external surface determination software within TIAMAT.

4.7 Coarsening 2-Elements

The 2-element coarsening strategy presented in this thesis, utilizes face sharing between elements to identify which neighbours to merge into an 8- to 26-node parent element. Once two neighbouring (sibling) elements are found to share a face (i.e. both elements share the same 4 corner nodes), they are identified for coarsening and undergo a number of validation checks to ensure they will create a valid element. If the two elements pass all the checks, they are merged. During merging, the eight unshared corner nodes of the two sibling elements become the corner nodes, while the four shared nodes become midedge nodes of the new parent element. This can be seen in Figure 4.3.

With all the nodes identified and ordered, the four midedge nodes are shifted so that they sit on a straight line between corner nodes. This was done for two reasons. First, it was believed that well-shaped coarsened elements were important for overall mesh quality. Second, the visualization software, LS-PrePost, does not show midedge nodes and therefore cannot show bent or curved edges. This can be very misleading for a user if the mesh appears visually different from the true mesh topology.

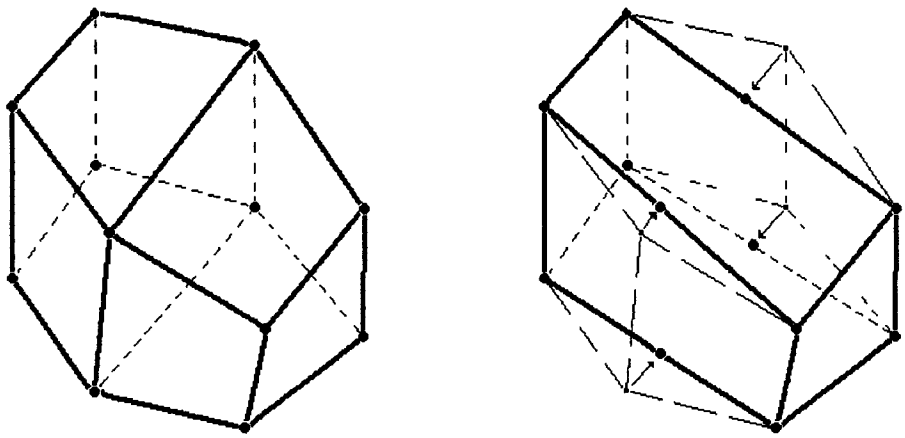


Figure 4.3: Merging of two elements. a) Two elements identified for coarsening b) Elements are merged and midedge nodes are shifted in-line with corner nodes

To identify elements for coarsening, a number of steps are taken to ensure the elements match up, create valid parent elements and do not deform the surface of the geometry. Figure 4.4 illustrates the steps developed for this 2-element coarsening scheme. The individual steps are described in detail in later sections, however Figure 4.4 includes references to figures which illustrate some of the checks being applied.

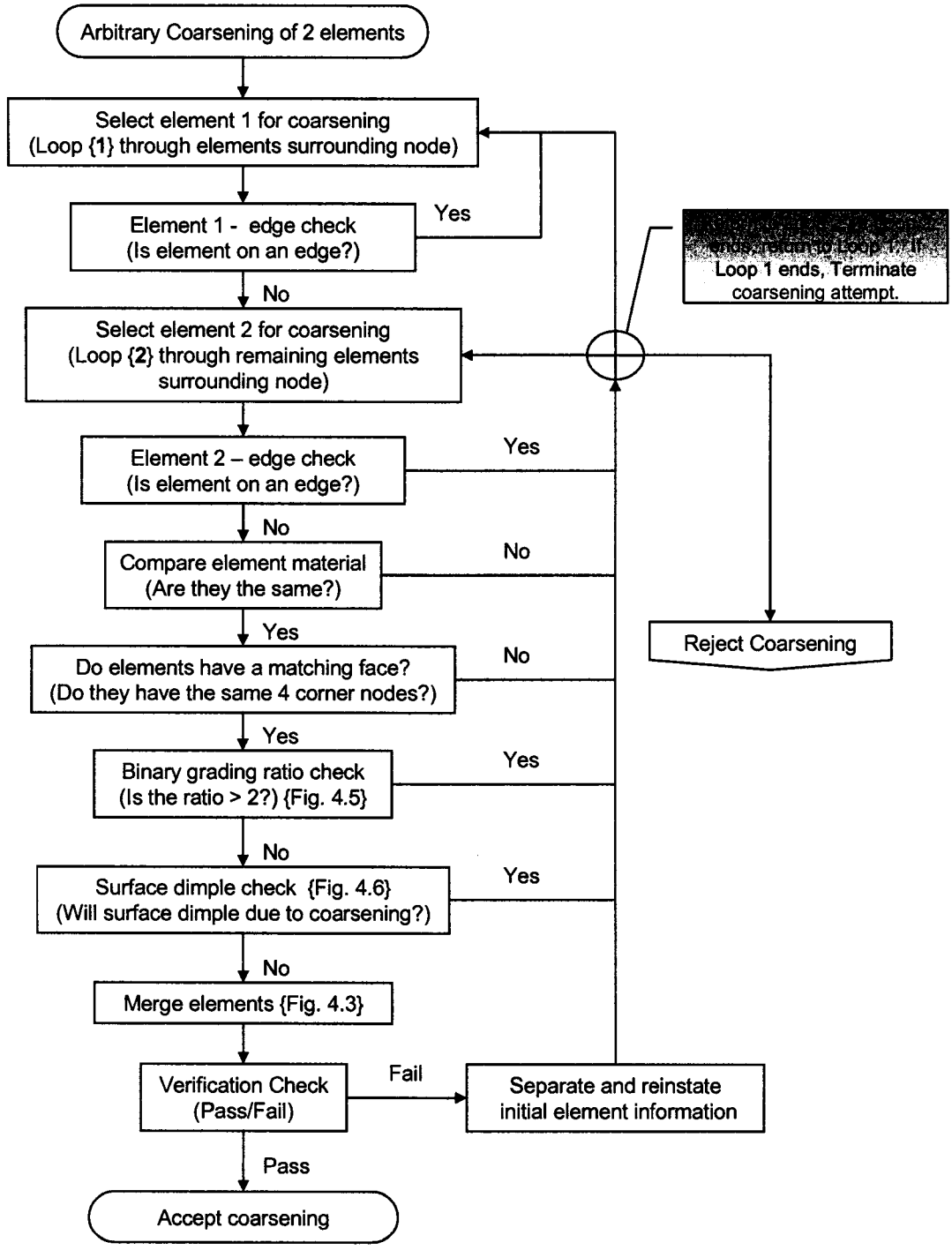


Figure 4.4: 2-element coarsening scheme

A nested loop was used to search through a list of all elements neighbouring the selected node. The adjacency relationships established earlier were used to construct the

list of neighbouring elements. The first loop selects element #1 from the list of neighbouring elements and checks if it resides upon an edge of the geometry. If it does, another element is selected from the list. If it does not, then loop #2 will begin by selecting element #2 from the remaining elements on the list. If element #2 is also found not to reside upon an edge, then the material types for both elements are compared.

The material type must match, since elements of different materials cannot be merged together. If the elements have the same material type, then the elements go through a face check. The face check will establish whether or not the two elements share a face. If they do not, loop #2 will select another element from the list. If the elements share a face, they are checked for binary grading. Binary grading refers to the mesh-grading ratio between neighbouring elements. The binary grading is limited to a ratio of 2-to-1, as shown in Figure 4.5, since the 8- to 26-node hexahedron only has room for one midedge node per edge. Elements are not allowed to merge if the new parent element requires more than one midedge node along a single edge to maintain the topology and connectivity of neighbouring elements.

For the next step, a dimple check is performed. The dimple check determines if any part of the boundary surface will be pulled into the interior due to the shifting of edge nodes during merging. Dimpling, as seen in Figure 4.6, tends to happen when merging elements share an irregular node on the surface of the mesh. If dimpling of the surface is found to happen, element #2 is rejected and another is selected. For those elements that do not dimple the surface, they will be merged into a parent element.

Once merged, a verification check is performed to ensure that the new parent element and all surrounding elements are not severely distorted due to the coarsening

process. The verification check looks for inverted or severely distorted elements such as those that have become wedge-shaped. If any of the elements are found to fail the verification check, the parent element is reset to the initial two sibling elements. If two elements are found to meet all the checks discussed, then the coarsening is accepted and the next stage of the mesh-management module is applied. If the first loop ends without finding two elements acceptable for coarsening, then coarsening around this node is rejected and the user is advised.

4.7.1 Element-Edge Check

The element-edge check subroutine is similar to the edge check subroutine discussed in Section 4.6, but operates slightly differently. The edge check subroutine determines if the selected node resides upon a boundary edge, whereas, this function determines if an element makes up part of a boundary edge. This function is necessary to ensure that edge elements neighbouring the selected node are not allowed to coarsen.

This subroutine uses two indicators to establish if the element resides along a boundary edge. The first indicator determines if the element has a free face. If the element does not have a free face, then it does not reside upon a boundary edge. If the element does have a single free face, then the unit normal vector of the free face is compared to those of the surrounding free faces. If the unit normal vectors differ by more than 20 degrees, it resides upon an edge.

The second indicator determines if the element has two or more free faces. If this condition is true and if any of the free faces share an element edge, then it resides upon an edge.

Any elements found to be part of a boundary edge are rejected and another element is selected from the list of neighbouring elements.

4.7.2 Material-Type Check

For the coarsening scheme to be practical, it had to have the functionality to handle meshes composed of different materials. Meshes with multiple materials require that each element material be tracked. This was done within the elementData array described earlier.

Multiple material types within a mesh adds boundaries, over which, elements may not coarsen. Only those elements composed of the same material may merge together.

The material check is a quick comparison of material codes between elements. If they do not match, element #2 is rejected and loop #2 starts another iteration of the process with a new element taken from the list of neighbouring elements.

4.7.3 Face Check

The face check is the principal function for identifying elements for 2-element coarsening. This function determines if the two selected elements share a face. If the elements do not share a face, they cannot be merged. The principle of face sharing used by this scheme, is based on common vertex nodes. If two elements contain the same four vertex nodes, they share a face. Although elements may partially share a face due to mesh grading (i.e. two or more smaller elements share the face of larger element), the scheme presented here will only recognize full face matches. Using vertex nodes to match full element faces reduces the complexity for merging elements. Partial face

matching will not only complicate the element merging process, but has a greater likelihood of creating a poorly shaped parent element.

4.7.4 Binary-Grading Check

As mentioned earlier, the binary-grading ratio check ensures that mesh-grading does not exceed a ratio of 2-to-1. This ratio will limit the number of nodes that can reside upon the edge of an element. It will also provide a smooth transition from fine to coarse regions of the mesh. The binary-grading limit is important since the 8- to 26-node hexahedral element can only support a single midedge node per edge. Binary-grading can be best illustrated using the 2-D mesh shown in Figure 4.5. In the initial mesh, element #1 borders element #3 and #4 along its bottom edge. To retain well-shaped elements and maintain connectivity, a midedge node has been applied to the bottom edge of element #1. The mesh-grading ratio along this edge is 2-to-1 (i.e. two smaller elements share the edge with a larger element). If element #1 and element #2 were to merge together, the shared nodes between them would become midedge nodes on the new parent element. Since element #1 already has a midedge node along its bottom edge, the addition of another would create a mesh-grading ratio of 3-to-1.

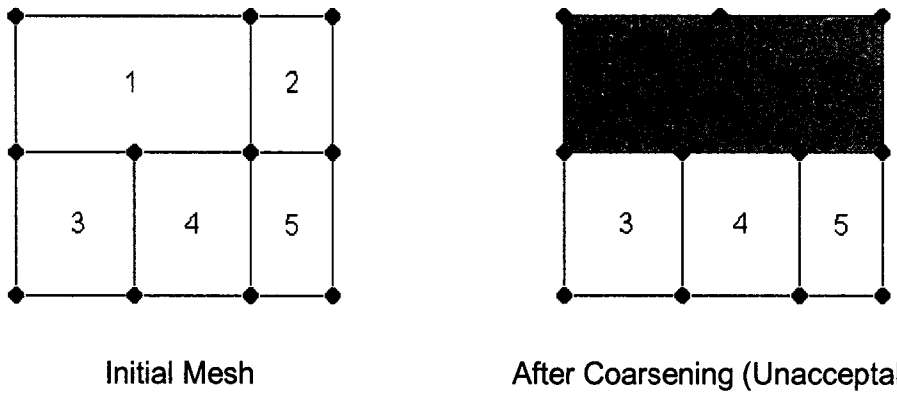


Figure 4.5: 2-D mesh containing 5 elements.

The binary-grading ratio check determines if either element has a node along the edges perpendicular to the shared face. If a midedge node is found, these elements are rejected for coarsening, returning for another iteration of loop #2 with a different element.

4.7.5 Surface-Dimple Check

The surface-dimple check was implemented after initial testing showed the mesh surface dimpled (i.e. surface nodes pulled towards the interior of the mesh) after coarsening near the surface. Surface-dimpling deforms the exterior face of a mesh, effectively changing the boundary of the domain. Dimpling breaks rule C5, and therefore, must not be allowed to happen. Figure 4.6 shows the top face of an object dimpled after coarsening.

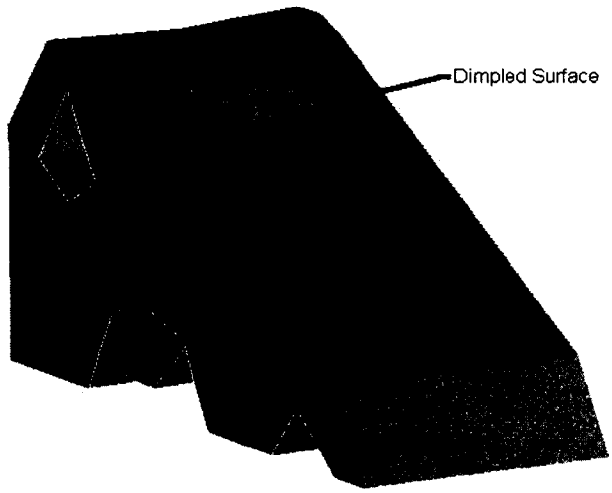


Figure 4.6: Surface dimpling.

Dimpling occurs when elements on or near the surface are coarsened. Experience has shown this occurrence happens particularly with unstructured, non-orthogonal meshes due to the number of irregular nodes residing within their domain. The dimpling effect is a direct result of the midedge node adjustment of irregular nodes on the surface during the merging process. Figure 4.7 (a) shows a 2-D example of a mesh with an irregular node on the surface. The irregular node is highlighted by a box.

There are two conditions under which dimpling will occur. The first condition occurs when a surface element merges with an interior element. If one of the shared nodes between the two elements resides upon the surface, then it will be shifted inward, dimpling the surface. Figure 4.7 (b) gives a simple illustration of this occurrence.

The second condition involves merging two interior sibling elements that share a node on the surface. Due to the nature of irregular nodes, an element can be interior to the domain, yet use one or two surface nodes as part of it topology. During coarsening, the edge node is adjusted inwards leaving a dimple in the surface. Figure 4.7 (c) illustrates the second condition.

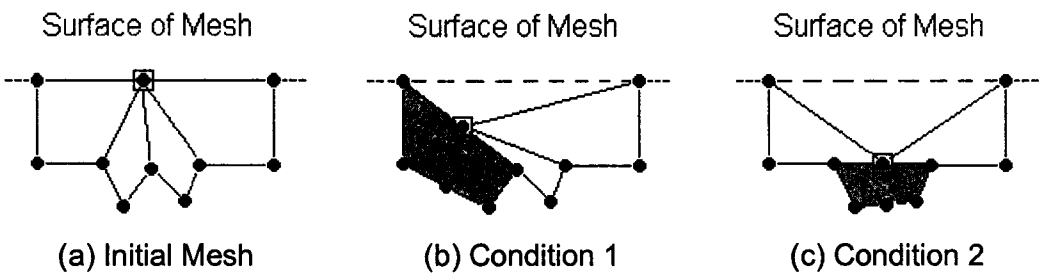


Figure 4.7: Dimpling of a 2-D surface.

As Figure 4.7 shows, dimpling not only deforms the surface, but also tends to severely distort the neighbouring elements.

To avoid dimpling, two checks are made against the elements identified for coarsening. Consider first the circumstances of Condition 1 (Figure 4.7 (b)). If a surface and interior element are identified for merging, the function will check if any of the four shared nodes reside on the surface. If any of these nodes do reside on the surface, then the merger of these two elements are rejected.

The second check considers the circumstances of Condition 2 (Figure 4.7 (c)). If two interior elements are identified for merging, the function will likewise check if any of the shared nodes reside on the surface.

4.7.6 Merging Elements

Once two elements have passed the previous checks, they will be merged into a parent element. The merging process involves the following 3 steps:

(1) Vertex, midedge and midface node ordering of parent element

(2) Backup (storage) of initial element information

(3) Midedge/midface node shifting

As described earlier, the merging process uses all the nodes from the original two elements to define the topology of the parent element. When defining the vertex, midedge and midface nodes of the parent element, the local ordering of these nodes becomes important to establish. Without correctly ordering the nodes, the basis functions will represent the wrong nodes and the parent element will be flagged as inverted by the verification check

McDill et al. [52] presented a rotation strategy to map each node from the sibling elements to their computational coordinates, determining their local node order in the parent element. Although this method is effective, it is believed to be computationally more intensive than the alternative method presented here. This alternative method uses the orientation of the sibling elements to define the local node order. The orientation of an element depends on the direction of the computational coordinate axis compared to the physical coordinate axis. Due to the nature of TTH-generated meshes, each element is oriented differently, making it necessary to define the difference in orientation between the sibling elements identified for coarsening. Figure 4.8 illustrates how the orientation may differ between elements.

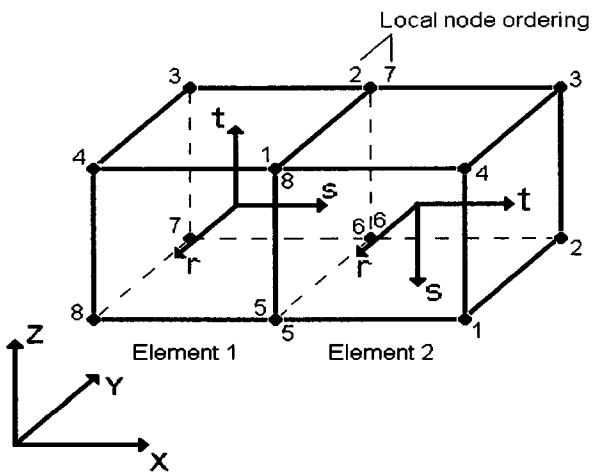


Figure 4.8: Local orientation of sibling elements

To establish a node order for the parent element, the node order from element #1 is used. The four unshared nodes on element #1 are extracted using a local face-node order list. Two face-node ordering lists are required. Face-node list #1 presents the local node order (counter-clockwise) used to define a face on an element, while Face-node list #2 presents local node ordering (clockwise) used to define an opposite face. When the lists are used in conjunction, the proper node order is established for opposite faces. In Figure 4.9, notice that face #1 is opposite to face #2, face #3 is opposite to face #4 and face #5 is opposite to face #6.

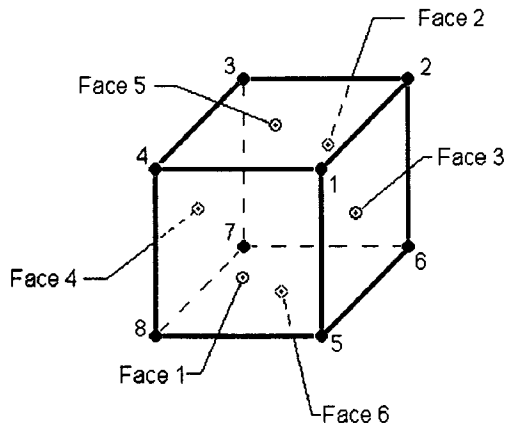


Figure 4.9: Element face order

The Face-node lists are shown in Table 4.1. The Face-node lists are shown in order of vertex nodes (position 1-4), edge nodes (position 5-8) and a single face node (position 9). Vertex nodes are mandatory for defining an element, but midedge and midface nodes will only appear when there is mesh-grading along an edge or face.

Table 4.1: Face-node local order lists for extracting local node order

Face	Face-Node List #1	Face-Node List #2
1	1, 4, 8, 5, 12, 20, 16, 17, 21	5, 8, 4, 1, 16, 20, 12, 17, 21
2	6, 7, 3, 2, 14, 19, 10, 18, 22	2, 3, 7, 6, 10, 19, 14, 18, 22
3	1, 5, 6, 2, 17, 13, 18, 9, 23	2, 6, 5, 1, 18, 13, 17, 9, 23
4	3, 7, 8, 4, 19, 15, 20, 11, 24	4, 8, 7, 3, 20, 15, 19, 11, 24
5	1, 2, 3, 4, 9, 10, 11, 12, 25	4, 3, 2, 1, 11, 10, 9, 12, 25
6	8, 7, 6, 5, 15, 14, 13, 16, 26	5, 6, 7, 8, 13, 14, 15, 16, 26

To determine the parent element local node order, the four unshared vertex nodes from element #1 are extracted using Face-node list #1. These nodes are used to define the top face of the parent element. The four unshared vertex nodes from element #2, extracted using Face-node list #2, define the bottom face.

For example, in Figure 4.8, face #3 of element #1 shares the same four vertex nodes with face #6 of element #2. Therefore, the top face of the parent element will be

defined by the unshared vertex nodes from element #1 (face #4), extracted in the order {3, 7, 8, 4}. The bottom face of the parent element will be define by the unshared vertex nodes from element #2 (face #5), extracted in the order {4, 3, 2, 1}. At this point, if these nodes are combined in their current order {Element #1: Element #2 | 3, 7, 8, 4, 4, 3, 2, 1}, they will create the twisted parent element seen in Figure 4.10.

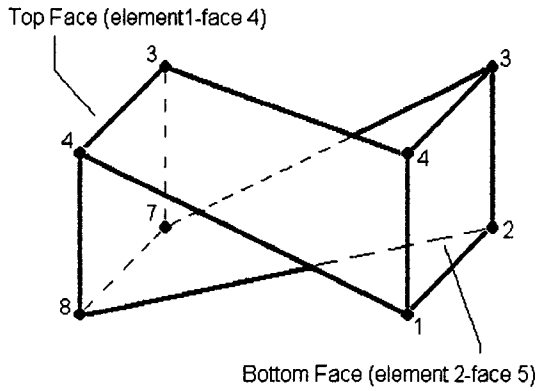


Figure 4.10: Twisted parent element

In order to straighten the parent element into a proper hexahedron, the bottom face has to be rotated clockwise. To computationally identify the degree of twist between faces, the difference in local node order between the shared faces is compared. From Face-node list #1, the vertex node order of face #3 (element #1 shared face) is {1, 5, 6, 2}. Likewise, the vertex node order of face #6 (element #2 shared face) from Face-node list #2 is {5, 6, 7, 8}. From Figure 4.8, it can be seen that local node #5 of element #2 is the same global node as local node #5 of element #1 (their local node number match is a coincidence). Therefore, if the first node (node #5) is selected from face #6 of Face-node list #2 and its position is compared to that of face #3 of Face-node list #1, the degree of rotation can be determined.

As seen from the difference in node order between the two lists, it is one position off. Each position relates to a 90-degree rotation difference between computational axes. Rotating the bottom face by 90-degrees means a shift in the position of the nodes by one space to the left. Therefore, the local order for the bottom face becomes {3, 2, 1, 4}, and the order for all eight vertex nodes of the parent becomes {Element #1: Element #2 | 3, 7, 8, 4, 3, 2, 1, 4}. Once local node ordering is established the global numbers for these local node numbers are placed within an array following this order. Once the vertex nodes are established for the parent, the unshared midedge and midface nodes are established in the same manner. These nodes will become nodes on the parent in the local node positions: 9, 10, 11, 12, 13, 14, 15, 16, 25 and 26.

The shared midedge and midface are applied in a similar way. The four shared nodes of element #2 become edge nodes on the parent and are positioned using Face-node list #1 (face #6). Their position must be shifted using the same degree of rotation established above.

The Backup step involves saving the element and node information of the two elements undergoing coarsening to another location. This must be done so that if the verification check rejects the merger of the two elements, their initial data may be restored. After backing up the data, the element data is replaced with the new parent element information established above. The parent element takes on the element id of the sibling with the lower id value. The second sibling information is removed during the re-numbering stage of the Mesh-Management Module.

The final step of the merging function deals with shifting the midedge (shared) nodes so that they reside upon a straight line between vertex nodes. This step only

affects distorted elements, since the edge nodes on an orthogonal element will already reside upon a straight edge between vertex nodes. The node coordinates of each midedge node are shifted to the middle ground between the vertex nodes bordering them. The following linear interpolation method was used to reposition the midedge nodes of a coarsened element.

$$X_{17+i} = (X_{1+i} + X_{5+i}) / 2 \quad (4.1)$$

$$Y_{17+i} = (Y_{1+i} + Y_{5+i}) / 2 \quad (4.2)$$

$$Z_{17+i} = (Z_{1+i} + Z_{5+i}) / 2 \quad (4.3)$$

where; i = 0, 1, 2, 3

X, Y and Z are global coordinates of nodes and subscript numbers refer to the local node number seen in Figure 4.11.

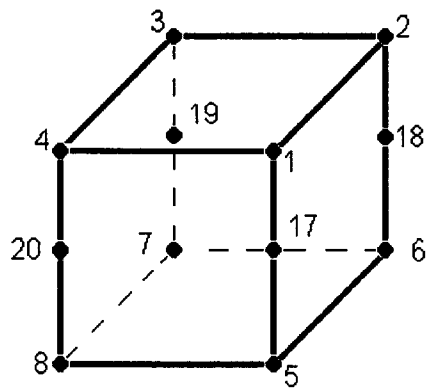


Figure 4.11: Local node order of parent element for the purpose of repositioning its midedge nodes (17, 18, 19 and 20).

Linear interpolation in physical space was used for the midedge node adjustment because it was a simple method to implement. An alternative method of using basis functions to find the physical coordinates of the midedge nodes was not implemented because the software had not been initially designed to easily accommodate drawing out basis function information.

If any shared nodes previous to coarsening existed as midedge nodes, they become face nodes on the parent element. Face nodes follow the same procedure for repositioning their coordinates, but instead of being centered between vertex nodes, they are centred between midedge nodes (17, 18, 19 and 20).

4.7.7 Verification Check

The verification check institutes a series of quality measurements and metrics to ascertain whether the coarsening operation meets the rules C2 and C3 discussed in Section 4.1. Initially, three quality metrics were used to evaluate element quality after coarsening transpired. The three metrics used were: aspect ratio (AR), skew angle and determinant of the Jacobian (det J). For efficiency, only the coarsened element and its neighbouring elements were examined, since they were the only elements to undergo any change in topology. The remaining elements within the mesh were not directly affected by the coarsening process.

The det J check was instituted first to catch any elements that may have become inverted during coarsening. To assess an element for inversion, the det J was calculated at all 8 Gauss points and at the centroid. As discussed in Chapter 2, if any det J values are found to be zero or less, then the element is considered to be inverted. If any of the elements evaluated are found to be inverted, the newly coarsened element is removed and the initial two sibling elements are restored.

Originally, the aspect ratio and skew angle metrics were meant to be used for measuring element quality and determining when elements become severely distorted, as in the case of wedge creation. It was hoped that skew angle measurements would detect wedge-shaped elements and, therefore, avoid their creation. It was later discovered that

the skew angle could not accurately detect all severely distorted elements. Since skew angles are a measure of the angle between computational axes mapped to physical space, they are most useful when the opposite faces of the hexahedron are parallel. If opposite faces are not parallel, such as in severely distorted elements (wedge-shaped, flattened, etc.) then the skew angles are found to be an inaccurate metric. Therefore, to detect wedge creation due to coarsening, an alternative method was used.

To detect elements of poor quality, the det J metric was used. This time, it was calculated at each node. Distorted elements, like wedges, will have values of det J equal to or less than zero at node locations where the angle between neighbouring faces of the element is zero. Measuring the determinant of the Jacobian at the node locations gave a good indication of which elements were severely distorted. If elements were not found to be inverted at their Gauss points but were found to fail the det J metric at their node locations, this signified that an element may have become wedge-shaped. This was not enough to determine if an element had become wedge-shaped, since testing of pre-coarsened TTH-generated meshes showed 12 to 27 percent of elements within a mesh failed this test. This meant that this test could indicate that an element had become wedge-shaped or that an element was misshapen prior to coarsening. If an element was severely distorted from the TTH mesh generation process, then the coarsening routine need not limit itself by disallowing any coarsening around such an element. Therefore, a second test was required to verify if the element had become wedge-shaped due to the coarsening process.

To test for wedge generation, a face angle scheme was used. It worked in a similar manner to the normal vector comparison routine within the edge check function.

The face angle scheme calculated the unit normal vectors for each face of the element and compared the directions of neighbouring face unit vectors. Wedge-shaped elements have two neighbouring faces with normal vectors that point in the same direction and therefore, have an angle difference of zero degrees. Figure 4.12 illustrates the unit normal vectors of two neighbouring faces on a wedge-shaped element. Additional information on the calculation of the normal vectors is presented in Appendix 2.

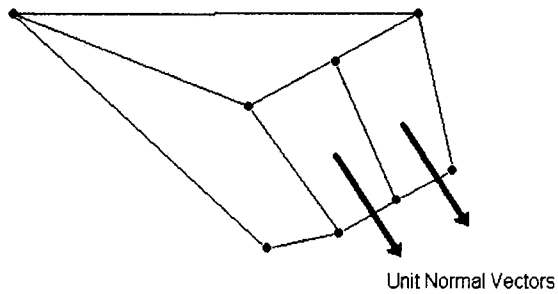


Figure 4.12: Wedge-shaped element with unit normal vector comparison

4.8 Redundant Node Removal

The redundant node removal step was implemented to remove any nodes that were no longer required for defining the topology of any elements. Elements require eight vertex nodes for a valid topology. Midedge and midface nodes are optional; i.e. they may not be required for outlining the topology of an element. They are used in linear hexahedra to principally define connectivity to graded neighbouring elements. Since connectivity is also established by vertex nodes, any node that does not define a vertex for any element is considered redundant and is removed. In a very refined mesh, redundant nodes can be costly, using computer resources that may be better utilized elsewhere.

Redundant nodes were identified using the adjacency relationships established earlier. Each node on the coarsened element was inspected to determine if it defined at least one vertex of any neighbouring elements. If not, it was removed from all element definitions.

4.9 Element Renumbering

The element renumbering stage was necessary to remove the old element information from memory and renumber the elements identification numbers after coarsening. The element numbering is important because TIAMAT requires element identification numbers to be continuous for input. TTH and the mesh management module were both created for the purpose of providing FEA model or mesh data that is compatible with TIAMAT.

Most FEA solvers are not able to work with the 8- to 26-node linear element used by TIAMAT for defining meshes. For other FEA solvers, it is possible to generate external constraint equations to deal with the optional nodes associated with mesh grading.

The element renumbering scheme was designed to remove the sibling element with the larger identification number. Since the coarsened element took over the element identity of the element with the smaller id, the second sibling element data had become invalid. To remove this element, all elements with id values greater than that of the second sibling element had to be shifted to a lower id value. This effectively removed the second sibling element and retained a sequential, continuous element numbering. The number of elements within the mesh was also reduced by one element.

Chapter 5: GITTH APPLICATION

The GITTH (GUI Interface to TTH) application was written in C (programming language) for a Red Hat (Linux) operating system. The primary mandate for GITTH was to operate within a Linux environment. Therefore, GITTH was developed using Linux libraries and compiled using the Red Hat (Version 9.0) C compiler. GITTH has not been tested on any other operating platform.

The GUI interface was designed using the Glade 2 User Interface Builder supported by GTK+ and GNOME libraries (both are Linux libraries written in C). Glade 2 is a Linux GUI development program available in Red Hat operating systems. The Glade software was used for developing GITTH because it provided a simple interface for designing a GUI and automatically coupled the necessary libraries to the application.

The development of the GITTH interface was driven by two main objectives. The first objective was to bring together the TTH application with all of its peripheral support programs and have them assembled into a single package. Since the TTH application and its peripheral software were all stand-alone programs, they were not easily navigated. Incorporating them into GITTH provided a platform where all the software was located together in a user-friendly environment.

The second objective was the facilitation of a mesh-management program. The mesh-management program was required to selectively coarsen hexahedral meshes created from the TTH software or any other applications that employs the same mesh format. For coarsening to be practical, GITTH was designed to provide a straightforward interface for using the coarsening function.

5.1 GITTH Interface

The interface for GITTH was designed to be user-friendly and provide a simple and clear approach to navigating between meshing functions. To this end, the interface was designed to group the TTH and coarsening functionality separately. The function calls were staged to follow the order of operation consistent with using this tool. By keeping all meshing functions on the main window, the interface was easy to navigate and meshing operations were easily followed.

The development of the interface had another issue to consider during conception of the coarsening process. As mentioned earlier, GITTH is not able to view a mesh itself. Therefore, a pre-processor was required to visualize TTH-generated meshes. The pre-/post-processor, LS-PrePost, was used to view the generated meshes and to support the mesh-management functions within GITTH. For GITTH to work together with LS-PrePost, the interface had to be small enough not to cover the LS-PrePost window during use. Therefore, the GITTH interface was designed to be narrow, so that it could act as a side panel without interfering with the visibility of the LS-PrePost window. The GITTH interface can be seen in Figure 5.1.

The top section of the interface contains the TTH application and its peripheral software. The peripheral software includes conversion functions that reformat mesh files into TTH readable formats or converts TTH data files into binary LS-PrePost readable files. The lower half of the interface contains the mesh-management functions. The layout of the function-calls follows the order in which mesh-generation and mesh-management generally would occur.

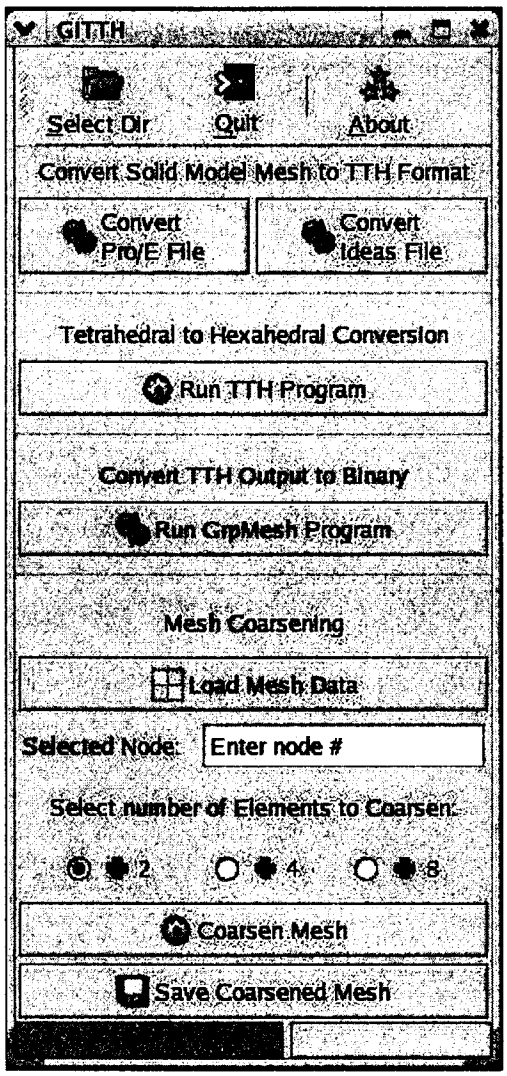


Figure 5.1: GITTH interface

5.2 TTH software

The top section of the GITTH interface contains the TTH application and its peripheral software functions. As mentioned earlier, TTH was designed to take tetrahedral meshes generated in Pro/Engineer or IDEAS and convert them into hexahedral meshes. Before TTH can be run, the Pro/Engineer or IDEAS data file must

be converted into a common format in which TTH can input the data. Therefore, there are two functions within GITTH to convert existing mesh files into TTH-readable format.

To view a TTH-generated mesh, the mesh data must be converted into binary and formatted into a LS-PrePost data file. This conversion is done by the GrpMesh function.

TTH and its peripheral software were developed before GITTH and therefore, were not designed to work within a GUI environment. They have been incorporated to work within GITTH, but have not been optimized for computer resource usage in a GUI environment.

5.3 Mesh-Management Module

The mesh-management module, located in the bottom section of the interface, handles the coarsening functionality. This section contains load/save mesh file buttons, a node input window and a coarsening size selection (2, 4 or 8 elements). Currently, only the 2-element coarsening algorithm has been developed. The Load/Save mesh buttons were implemented so that mesh data could be loaded or saved at any time.

The coarsening functionality was designed to work in conjunction with the commercial program LS-PrePost. GITTH and LS-PrePost are not directly linked and cannot inter-communicate. Instead, LS-PrePost is used as a reference tool, where areas to coarsen are identified within LS-PrePost by node numbers. The node numbers are then manually entered into GITTH to coarsen around that selected node.

To view a mesh after coarsening has transpired, it must be saved to file and then converted to binary using the GrpMesh button. Once in binary format it may be opened by LS-PrePost. This is not the most efficient method for coarsening, but was necessary

since there was no way to directly update meshes viewed within LS-PrePost at this time. By keeping GITTH and the mesh visualization software separate, it does allow the option of moving to other pre-processors in the future.

Chapter 6: TESTING AND VERIFICATION

This chapter details the validation testing performed on the coarsening software and presents observations on the mesh quality for the region surrounding the coarsened element. A number of test cases are presented to visually demonstrate 2-element coarsening and measure the quality of the coarsened region. The verification subroutine is then examined and optimized for detecting and avoiding the creation of wedge-shaped elements. Finally, a sample FEA validity test for coarsened meshes using the TIAMAT solver is considered.

6.1 Test Cases

A number of test cases were used to measure the performance of the coarsening software and give visual validation of the coarsening scheme. The test cases were chosen to showcase both orthogonal and non-orthogonal mesh coarsening.

The first two cases presented, are the plate and the cylinder. Both are comprised of well-shaped, orthogonal hexahedral elements. These cases demonstrate a minimum standard for coarsening. The coarsening algorithm must be fully proficient in coarsening good quality meshes, otherwise it would be impractical for coarsening non-orthogonal meshes.

The plate, shown in Figure 6.1, is a flat, one-element thick mesh. It is meant to illustrate coarsening in a single plane without the need for edge-node adjustment. The cylinder [12], seen in Figure 6.2, is a graded mesh with a curved boundary. It illustrates coarsening of graded meshes with minimal edge-node adjustments.

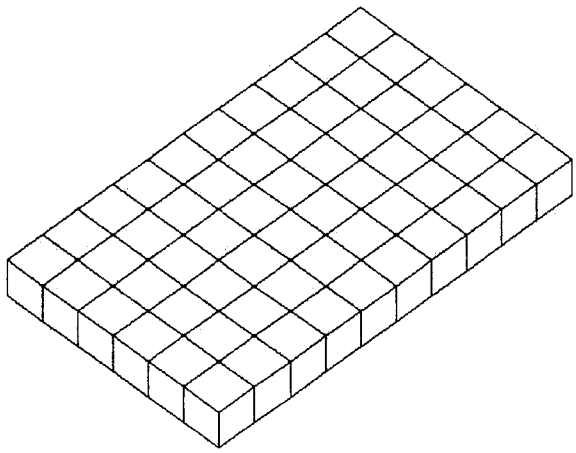


Figure 6.1: Plate

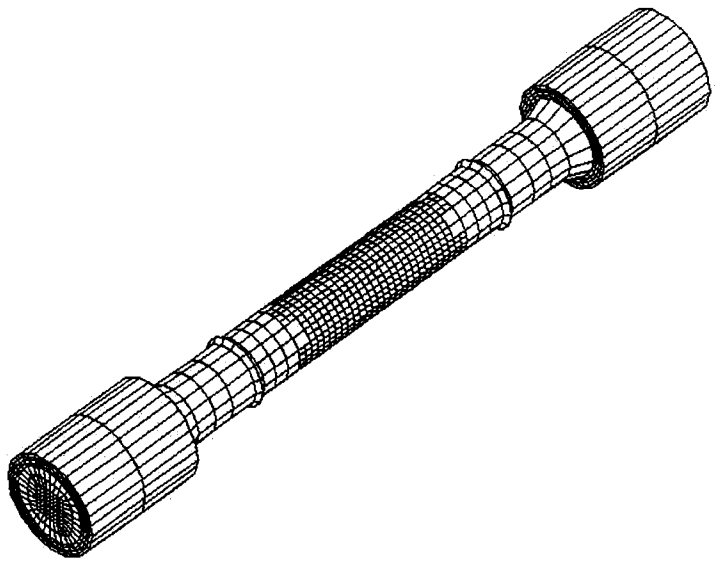
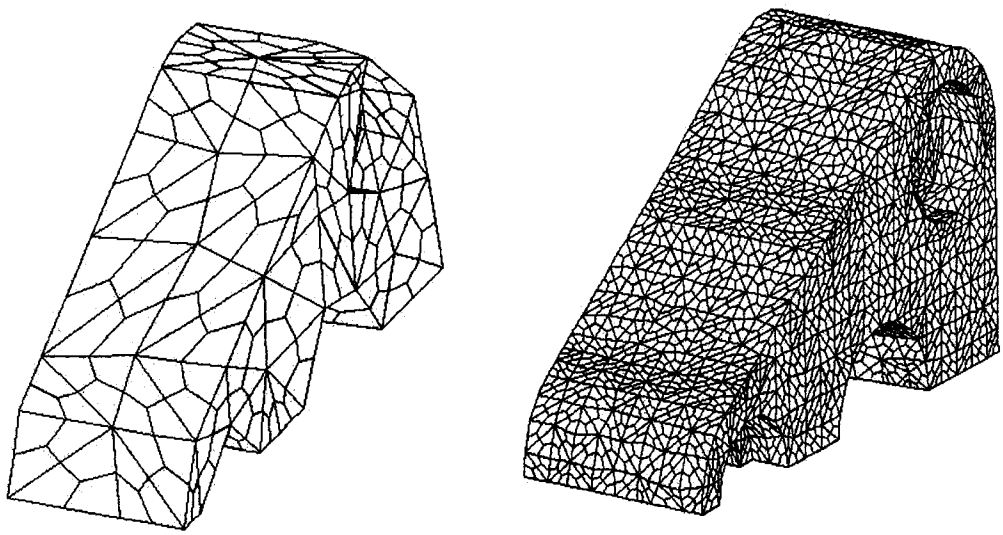


Figure 6.2: Cylinder [12]

The next two cases illustrate coarsening of TTH-generated meshes. These cases, shown in Figure 6.3, are coarse and fine mesh instances of a stationary platen model [6]. Due to the non-orthogonal nature of TTH-generated meshes, viewing coarsened elements within the interior of the mesh is difficult. Therefore, visual observations of coarsening will only be shown at the surface.



(a) Coarse Mesh (b) Fine Mesh

Figure 6.3: Stationary Platen cases [6]

Table 6.1 gives an overview of the initial mesh density for these test cases.

Table 6.1: Initial test case mesh densities

Part Name	Number of Elements within Mesh	Number of Nodes within Mesh
Plate	60	154
Cylinder	4,848	6,193
Stationary Platen (Coarse)	1,424	1,924
Stationary Platen (fine)	32,716	38,710

6.2 Mesh Quality of Elements in Coarsened Region

To measure the quality of elements within the coarsened region, four metrics have been used. These metrics are the: maximum skew angle, maximum aspect ratio, determinant of the Jacobian (det J) and a distortion parameter (DP).

The skew angle, aspect ratio and determinant of the Jacobian were discussed in Chapter 2. The distortion parameter [14], gives a numerical value for the overall degree

of distortion of an element. The distortion parameter (DP) for a hexahedron is found using the following equation:

$$DP = [(\text{min. det } J)_{r,s,t} / \det J_{0,0,0}] \quad (6.1)$$

where,

min. det J is the minimum value of the determinant of the Jacobian found at the vertex nodes,

$r, s, t = -1$ or 1 (node coordinates within computational space)

The distortion parameter measures element distortion through a comparison of the minimum det J found at the vertex node coordinates (r, s, t) to the det J measured at the centroid. Essentially, the DP value indicates the degree of deformation of a hexahedron compared to that of a parallelogram. The following DP ranges give a rough indication of element distortion:

DP = 1; no distortion,

0.2 < DP < 1; some distortion,

0 < DP <= 0.2; excessive distortion,

DP <= 0; severe distortion.

The distortion parameter has been applied to measure the performance of the coarsening algorithm in addition to the other quality metrics in order to determine their applicability for use as distortion control metrics. During the testing phase, it was found that the skew angle and aspect ratio were not as reliable as had been expected for elements that had undergone significant distortion.

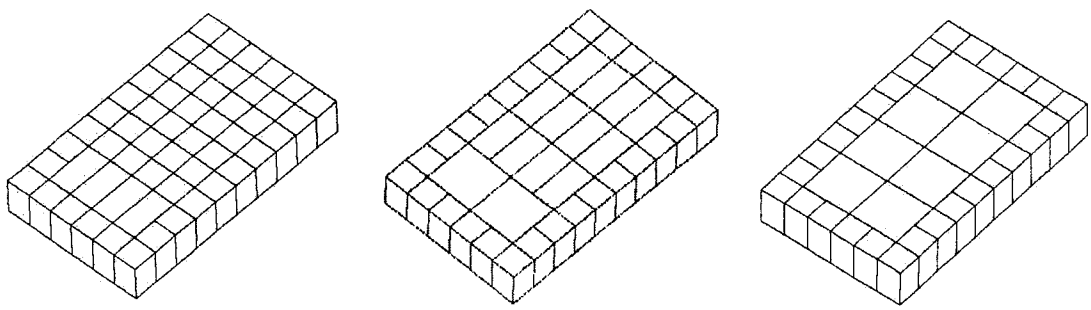
The skew angle metric requires an element to have a parallelogram shape to give an accurate measurement. As an element deviates from the parallelogram shape, the skew angle becomes less accurate.

The aspect ratio was found to be a poor indicator for detecting the types of distortion seen in TTH-generated meshes, such as wedge-shaped element creation.

The distortion parameter has not been formally implemented within the software. It has been used principally as a reference for verification testing.

6.2.1 Plate

The plate case verified the performance of the coarsening algorithm over a perfectly orthogonal mesh without any irregular nodes. Figure 6.4 shows various levels of coarsening within the plate. It can be seen that the boundary elements along the outer edge have not been coarsened. This is due to the edge rule, in which elements along an edge are not allowed to coarsen. Since the elements were all initially orthogonal, edge-node adjustment was not necessary during coarsening. Without the need to adjust edge-nodes, the coarsening algorithm did not cause any distortion to the surrounding area.



(a) 56 Hex, 148 Nodes (b) 42 Hex, 130 Nodes (c) 36 Hex, 118 Nodes

Figure 6.4: Coarsening elements within plate

Table 6.2 summarizes the changes in skew angle and distortion parameter before and after all the coarsening steps. The coarsening shown in Figure 6.4 (c), took 24 coarsening steps.

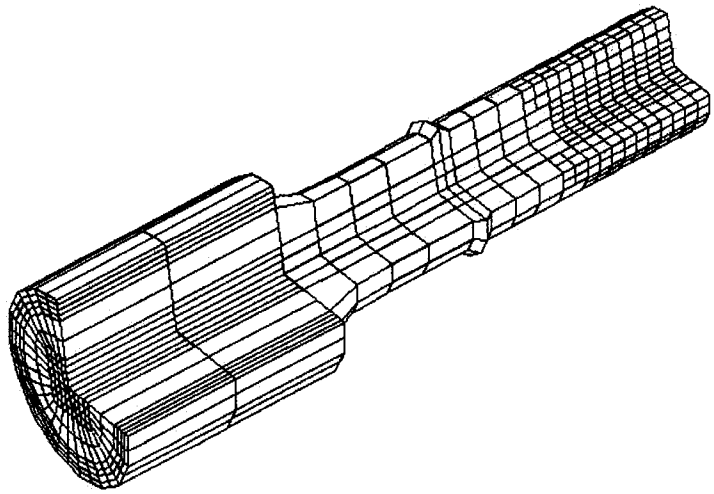
Table 6.2: Summary of quality metrics for plate case

Elements		Average Skew Angle (Degrees)	Average Distortion Parameter
Coarsened Elements		90°	1
Neighbouring Elements	Before coarsening	90°	1
	After coarsening	90°	1

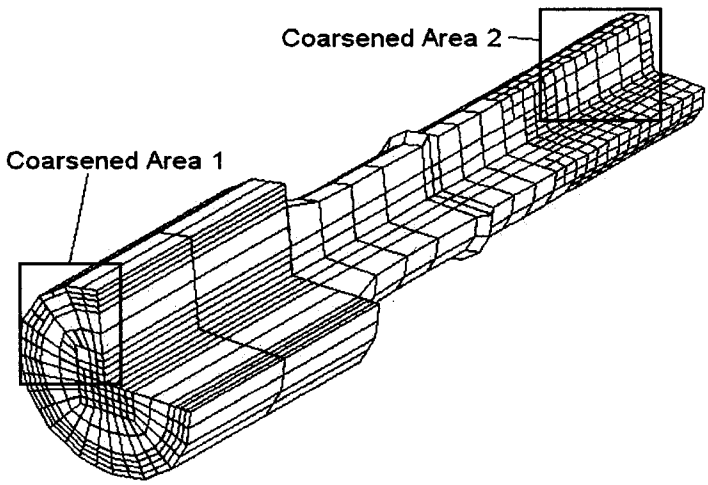
As seen in Table 6.2, the average skew angle and distortion parameter values for all elements within the coarsening region did not vary over the 24 coarsening steps. The aspect ratio remained constant as well.

6.2.2 Cylinder

The cylinder case presented a graded mesh with a mix of orthogonal and slightly distorted elements. This case has a few irregular nodes, located in areas where the orthogonal elements border slightly skewed elements. Due to the curvature of the exterior boundary of the cylinder, only the interior elements could be meshed. As seen in Figure 6.5 (a), the cylinder end with the larger diameter contains hexahedral elements with large aspect ratios. These elements were initially generated with large aspect ratios and should not be attributed to the coarsening algorithm presented in this thesis.



(a) Section view of Cylinder



(b) Coarsened: 4792 Hex, 6170 Nodes

Figure 6.5: Coarsening elements within cylinder.

The two areas shown in Figure 6.5 (b), illustrate coarsening of orthogonal and slightly skewed elements. The central elements within the cylinder are orthogonal, while the outer elements in Area 1 are slightly skewed due to the curvature of the outer boundary. There were 56 coarsening steps taken in Figure 6.5 (b). Table 6.3 summarizes a selection of the coarsened element quality metrics.

Table 6.3: Summary of coarsened element quality metrics

Coarsening Step	Coarsened Area	Aspect Ratio	Skew Angle (Degrees)	DP
1*	1	16	90°	1
3*	1	16	90°	1
6	1	6.5	72.3°	0.70
8	1	7.1	24.4°	0.95
12	1	5.8	40.9°	0.93
30*	2	2	90	1
56*	2	2	90°	1

The large aspect ratios seen in Table 6.3 are due to the initial mesh composition. Most of the elements in Area 1 are elongated, creating the large aspect ratios. Coarsening steps 1, 3, 30 and 56 show two orthogonal elements merging together into another orthogonal element. The other steps were of two slightly skewed elements coarsened together. The skew angle metric shows a large deformation, a change from 90° to 24.4°, on step 8. Although the DP values indicate slightly deformed elements created in steps 8 and 12, the skew angles indicate poor quality elements. A visual illustration of the coarsened element from step 12 is shown in Figure 6.6.

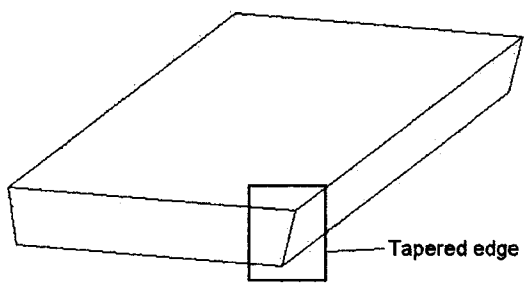


Figure 6.6: Coarsened element from step 12

As observed in Figure 6.6, the coarsened element does appear to be tapered slightly, but not significantly deformed. The tapered vertical edges are skewed slightly,

* The two sibling elements coarsened together were both initially perfectly orthogonal.

measuring an angle of 78.8° at the top corner in the highlighted section. The tapered angle indicates that the skew angle is not giving a true representation of certain elements.

The change in neighbouring element quality metrics due to coarsening can be seen in Figure 6.7. Only a single coarsening step caused a change in skew angle, AR and DP of elements neighbouring the coarsened element. The change in quality metrics is small, with less than a 2.5% maximum change in DP, and maximum changes of less than 1.5% for AR and skew angle. Each coarsened element had 20 to 30 neighbouring elements.

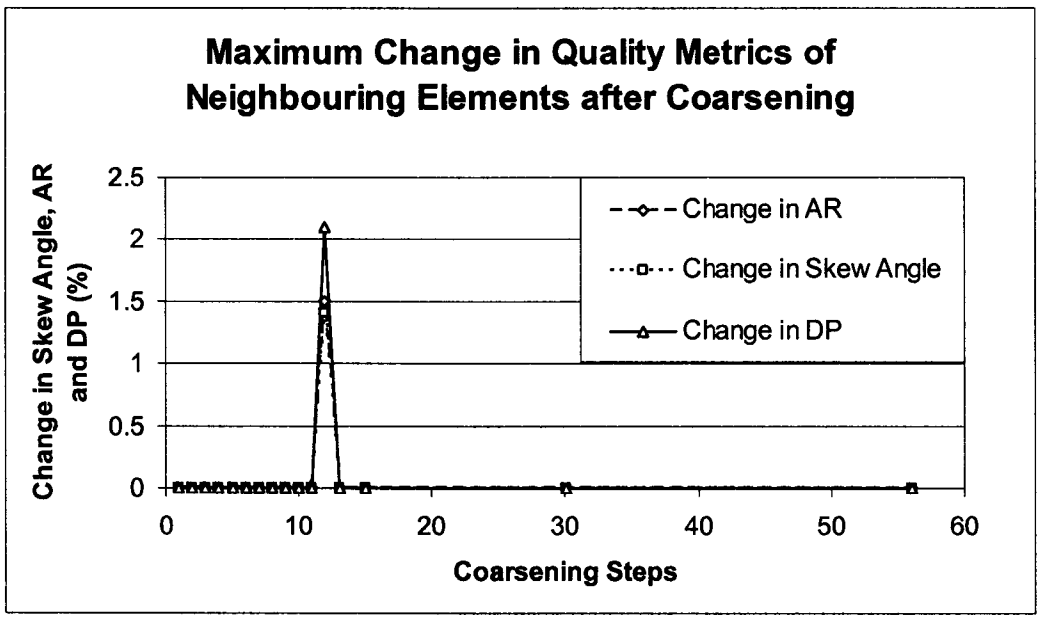
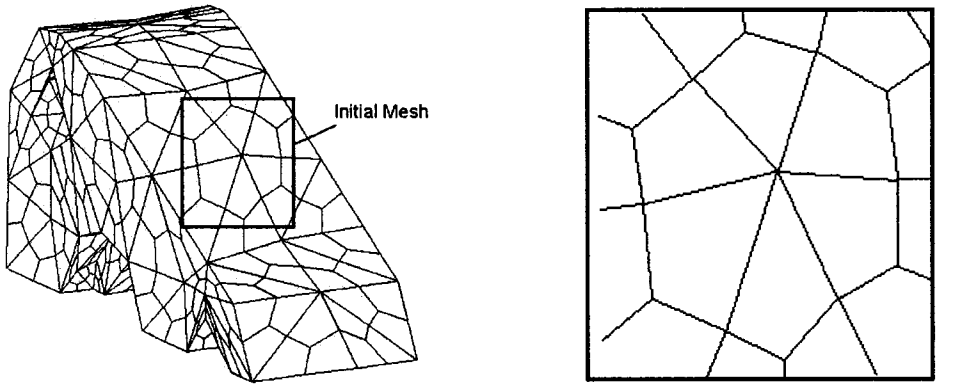


Figure 6.7: Maximum change in quality metrics of neighbouring elements after coarsening

Figure 6.7 shows the maximum change in quality metrics of neighbouring elements for each coarsening step. The low level of metric distortion within this graph signifies that good quality meshes with relatively few irregular nodes are not significantly affected by the coarsening software.

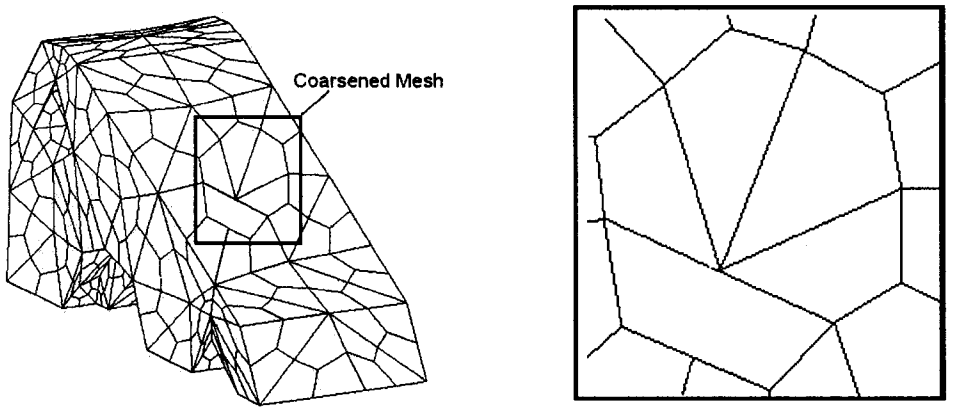
6.2.3 Stationary Platen (Coarse)

The coarse Stationary Platen case provides the first test for verifying the coarsening software on a TTH-generated mesh. This coarse TTH-generated mesh provides a look at coarsening elements of poor quality and the effect of irregular nodes on 2-element coarsening.



(a) Initial Mesh (b) Initial Mesh – Enlarged View

Figure 6.8: Stationary Platen (Coarse) – Initial Mesh



(a) Coarsened Mesh (b) Coarsened Mesh – Enlarged View

Figure 6.9: Stationary Platen (Coarse) – Coarsened Mesh

Figure 6.9 illustrates an example of 2-element coarsening within a TTH-generated mesh. The effect of coarsening on the surrounding region will vary depending on the severity of nearby irregular nodes and the degree of distortion of local elements. The

severity of an irregular node is measured by the number of elements connected to it. The greater the number of elements neighbouring a node, the greater the difficulty in coarsening around that node.

Since the mesh is made up of non-orthogonal elements, midedge node adjustment is required during the merging process. As observed from comparing Figure 6.9(b) to Figure 6.8(b), some of the neighbouring elements are visibly altered when the edge-node is adjusted. The quality metrics for the coarsened element from Figure 6.9 are shown in Table 6.4.

Table 6.4: Quality metrics for a coarsened element from Stationary Platen (Coarse)

Coarsened Element	Aspect Ratio (AR)	Skew Angle (Degree)	Distortion Parameter (DP)
Step 1	1.38	134°	0.52

The skew angle, aspect ratio and distortion parameter shown in Table 6.4 indicate the element is distorted, but not beyond the tolerances laid out in early sections (i.e. 45° to 135° limit for Skew Angle, maximum 2.5 AR and DP > 0). Visually, the coarsened element has a tapered shape, but appears to have a better shape than some of its neighbouring elements.

The neighbouring element skew angles and DP metric deviation due to coarsening step 1 are summarized in Figures 6.10 and 6.11. The aspect ratio deviation is not shown below since it was found to deviate by small amounts. This is expected since the edge-node adjustment should not cause a large change in the aspect ratio of neighbouring elements.

The trend in Figure 6.10 shows the coarsening software causes a change in the skew angles of the majority of the neighbouring elements. The skew angles appear to move farther away from the orthogonal (90°) profile.

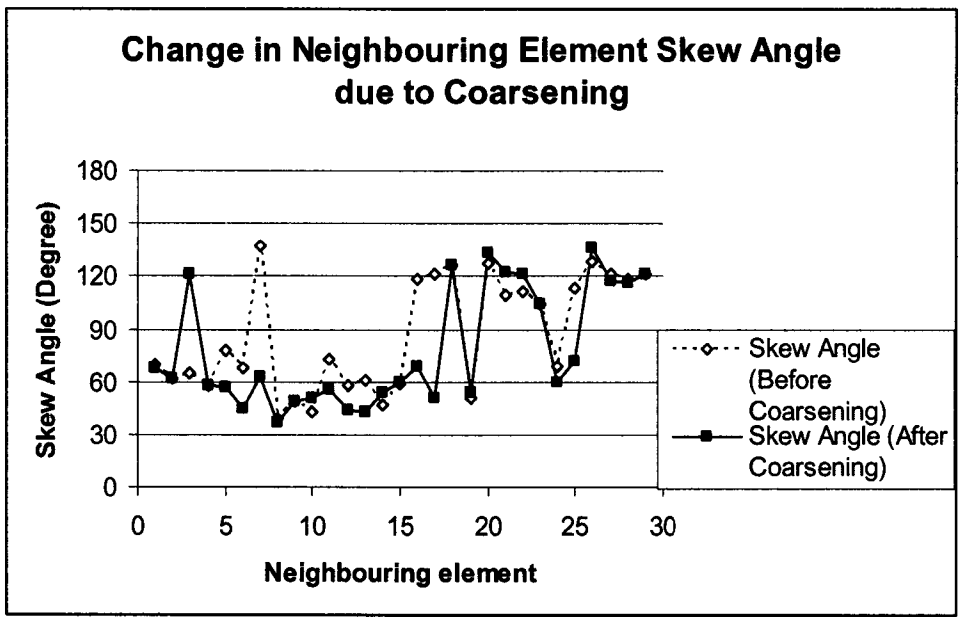


Figure 6.10: Skew angle deviation due to coarsening

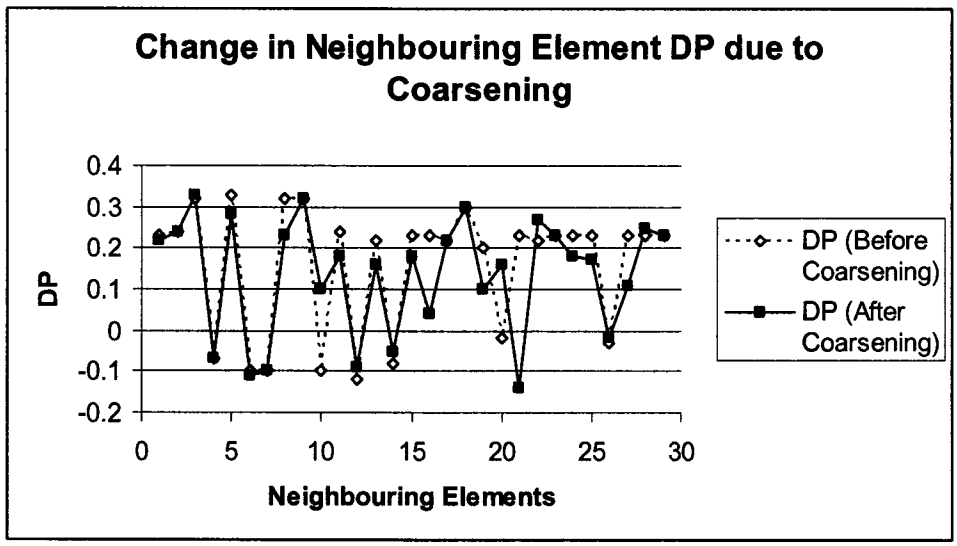


Figure 6.11: Distortion parameter deviation of Stationary Platen (Coarse)

The distortion parameter values shown in Figure 6.11 also show the majority of neighbouring elements becoming slightly more distorted. A number of the neighbouring elements have DP values measuring less than zero. This indicates that these elements are severely distorted. The skew angles for these same elements should show values approaching the limits of 0° or 180°. Instead, they have skew angles within or close to the established skew angle tolerance (45° to 135°). This confirms that the skew angle calculations are not reliably measuring the shape of highly distorted elements.

Figure 6.11 also illustrates the initial condition of the mesh in this region. All of the neighbouring elements have initial DP values under 0.4. This degree of distortion within the mesh makes it difficult to merge two elements together during coarsening without affecting the surrounding elements. Of the elements with DP values below zero, only a single case was caused by the coarsening software.

The trend of mesh distortion due to the coarsening software is seen in Figure 6.12. This graph illustrates the average change (%) in DP of neighbouring elements for each coarsening step. The trend in skew angle deviation over several coarsening steps is not illustrated, since skew angles have been shown to provide poor quality metrics for meshes with this level of distortion. The coarsening steps were localized to a specific area, with a few steps centralized around the same nodes. As observed, each coarsening step causes a degree of deterioration to the surrounding mesh. The average DP value is found to change, mainly in the range of 0 to -30%. A degree of element quality deterioration after coarsening was expected due to the midedge node adjustment.

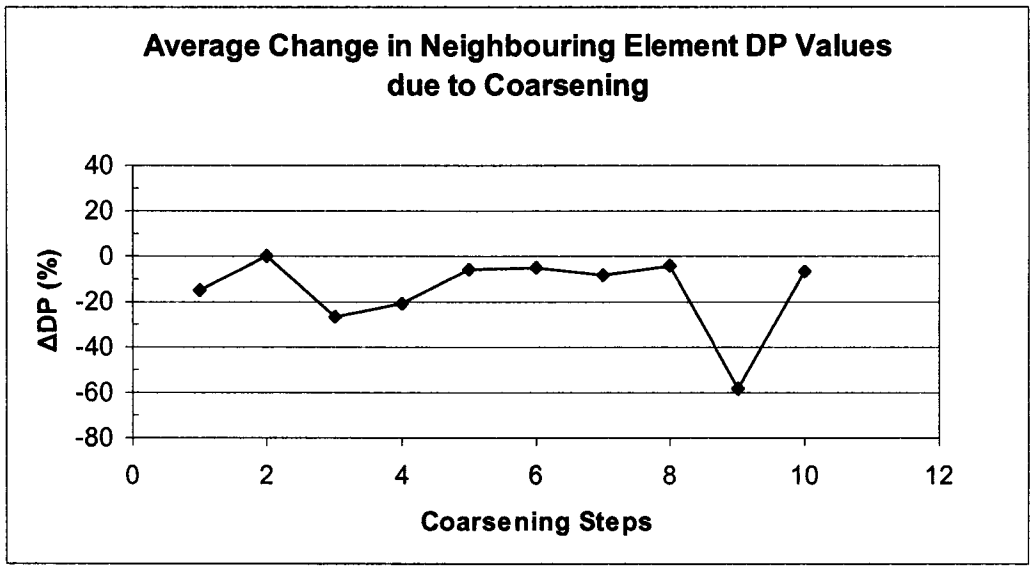
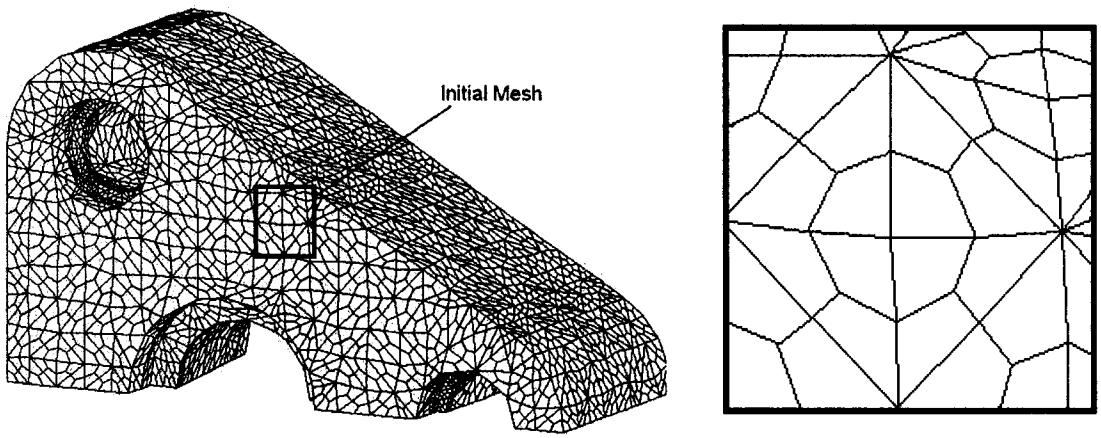


Figure 6.12: Trend in local mesh distortion for Stationary Platen (Coarse)

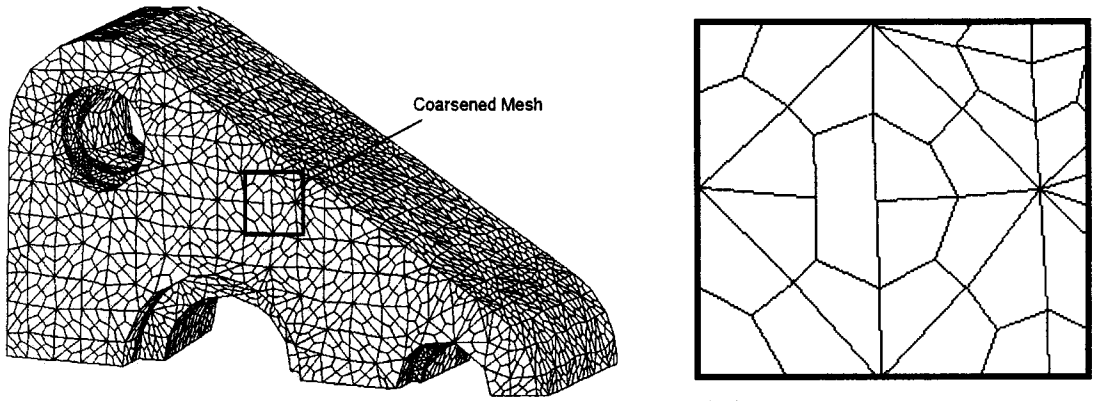
6.2.4 Stationary Platen (Fine)

The fine Stationary Platen case was studied to ascertain if a finer TTH-generated mesh provided a better initial mesh base for 2-element coarsening.



(a) Initial Mesh (b) Initial mesh – Enlarged View

Figure 6.13: Stationary Platen (Fine) – Initial Mesh



(a) Coarsened Mesh (b) Coarsened Mesh – Enlarged View

Figure 6.14: Stationary Platen (Fine) – Coarsened Mesh

Figure 6.14 illustrates a single coarsening step within the fine Stationary Platen case. Once again, the edge-node adjustment was observed to affect neighbouring elements. The quality metrics of the coarsened element are shown in Table 6.5.

Table 6.5: Quality metrics for a coarsened element from Stationary Platen (Fine)

Coarsened Element	Aspect Ratio (AR)	Skew Angle (Degree)	Distortion Parameter (DP)
Step 1	2.5	104.9°	0.47

Although the aspect ratio and skew angle seem reasonable for the coarsened element shown in 6.14 (b), the distortion parameter indicates a moderate amount of distortion. The distortion of this coarsened element appears to be similar to that of the coarsened element studied in Section 6.2.3. The visual appearance of both coarsened elements is similar.

The distortion parameter deviation of neighbouring elements is shown in Figure 6.15. The initial mesh distortion appears to be more consistent with the fine mesh case than that of the coarse mesh case. Similar to the coarse Stationary Platen case, the coarsening software causes deterioration of the DP values for the elements affected by

the edge-node adjustment. In this case, coarsening software did not cause any neighbouring elements to become severely distorted.

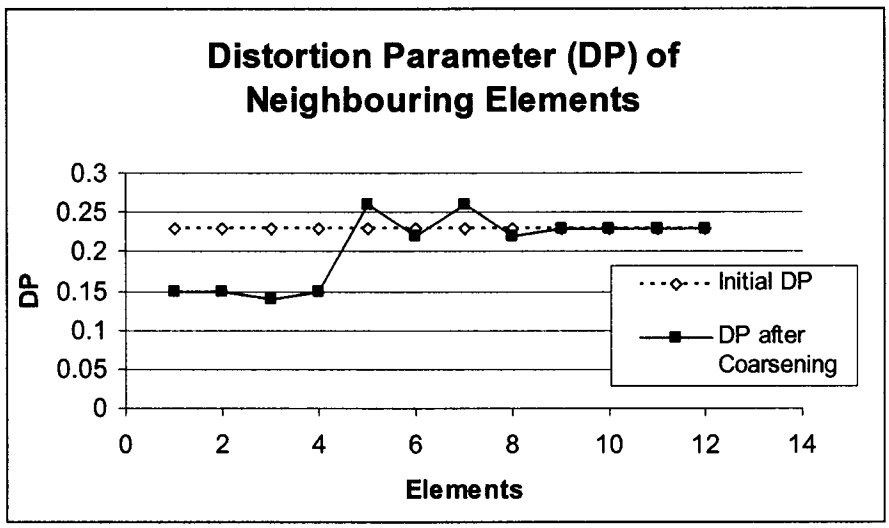


Figure 6.15: Distortion parameter deviation of Stationary Platen (Fine)

The trend in mesh distortion of neighbouring elements in the fine Stationary Platen is illustrated in Figure 6.16. The average deviation in DP values of neighbouring elements is mapped over 10 coarsening steps.

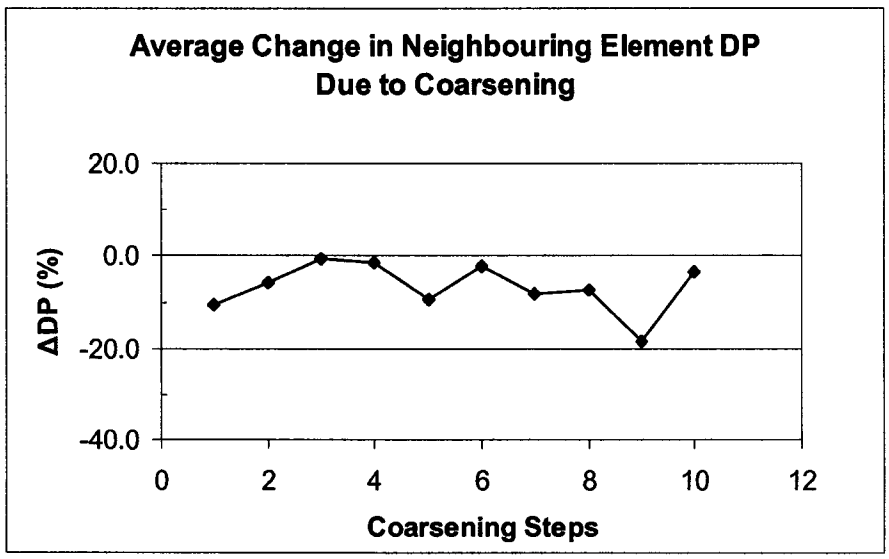


Figure 6.16: Trend in local mesh distortion for Stationary Platen (Fine)

The trend shown in Figure 6.16 shows an average deterioration to the quality of neighbouring elements. The range in deterioration to the fine mesh appears to be less than that of the coarse mesh case shown above. This may be due to the greater uniformity of distortion found in the fine Stationary Platen mesh.

It was found that the coarsening time lasted a few seconds for each coarsening step. The time it took to coarsen around a node is small and therefore can be considered insignificant to the overall mesh generation operation. It should be noted that as the mesh size increase, so does the coarsening time. For the Stationary Platen (fine) it took 1-2 seconds for coarsening on a Linux platform with a 2.8 GHz CPU processor. Larger meshes will take longer to coarsen, but the coarsening time will still remain in the range of seconds.

6.3 Wedge Element Avoidance

Wedge avoidance was designed into the verification subroutine to prevent the coarsening software from forcing neighbouring elements into a wedge-shape. As described in Chapter 3, the edge-node adjustment of a coarsened element causes certain neighbouring elements to become wedge-shaped. Wedge creation occurs predominantly when two sibling elements from the same parent tetrahedron coarsen. An example of wedge creation is shown in Figure 6.17.

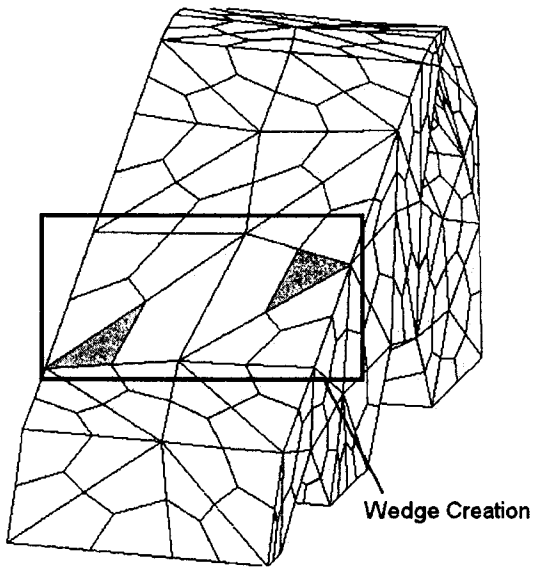


Figure 6.17: Wedge-shaped element creation

Two steps were taken to detect and prevent wedge-shaped elements from forming. Step 1 involved calculating the determinant of the Jacobian for each of the neighbouring element at each of their vertex nodes. Wedge-shaped elements have det J values equal to zero at the nodes where neighbouring faces have become planar. Although this step detects wedge-shaped elements, it also detects any severely distorted elements generated by the TTH software. Figure 6.18 illustrates the percentage of severely distorted elements found within various TTH-generated meshes. Coarse meshes have a larger percentage of severely distorted elements, with the number declining over finer meshes. The number of distorted elements within a mesh was also dependent on the complexity of the mesh geometry.

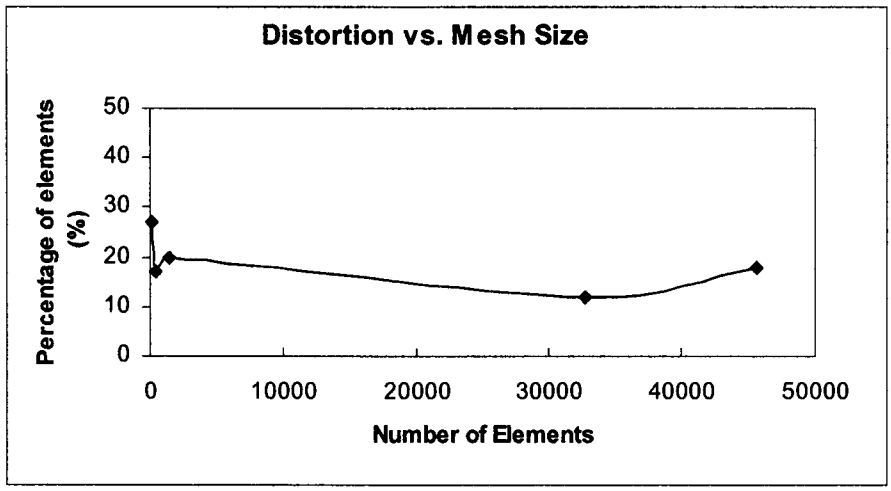


Figure 6.18: Percentage of severely distorted elements within TTH-generated meshes

The number of severely distorted elements within the original TTH-generated mesh made it difficult to detect when the coarsening software created a wedge-shaped element. Therefore, Step 2 was implemented to determine which severely distorted elements were created by the coarsening software.

Step 2 involved comparing the unit normal vectors of each face on a severely distorted hexahedron. When two neighbouring face normal vectors on the same element, are found to point in the same direction, the element is considered to have become wedge-shaped.

A study on the failure rate for detecting wedge elements based on the tolerance between normal vector angle differences is seen in Figure 6.19. A sample of 50 trials was conducted on the fine Stationary Platen mesh for each face angle tolerance case. The face angle tolerance is the allowable difference in normal vector directions for detecting in-plane neighbouring faces.

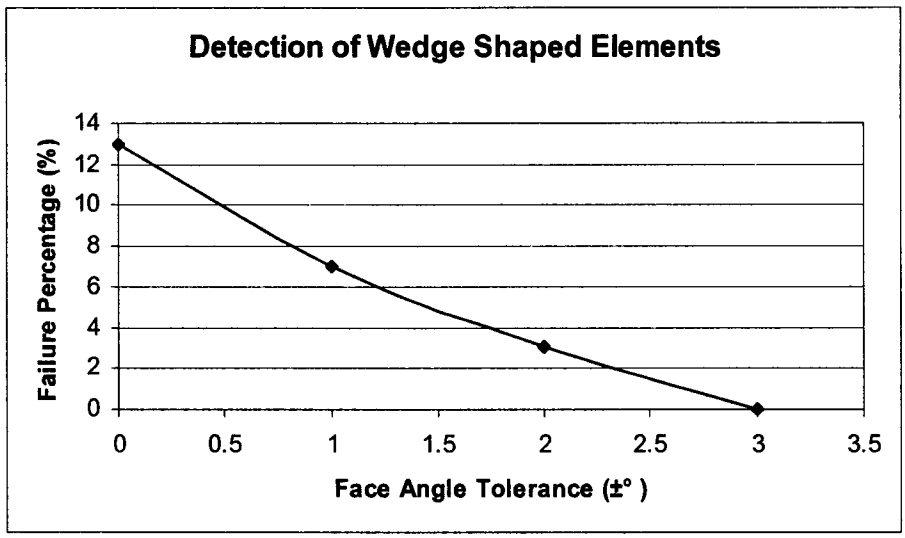


Figure 6.19: Wedge element detection failure versus face angle tolerance

It was discovered during testing that not all nodes on a face were in-plane. This meant the normal vector would deviate slightly from the expected direction. Testing showed that a ±3° face angle tolerance would reduce the detection failure rate to approximately 0% and this value was implemented in the software.

6.4 TIAMAT Verification

The coarsening software has been shown to work on orthogonal and non-orthogonal meshes, but to this point, has not been shown to create a valid mesh. To verify the coarsening algorithm, a coarsened TTH-generated mesh was run through TIAMAT. The mesh was used for a simple elastic stress analysis using a single point load source.

The point of the analysis was not to determine the accuracy of the coarsened mesh, but to ensure the coarsening algorithm does not create invalid elements. The mesh

was composed of 398 hexahedral elements, 595 nodes and had undergone 10 coarsening steps in various regions within the mesh.

The analysis ran successfully and demonstrated that the coarsening software produced valid mesh information. Further study will be required to assess the overall capability of the tetrahedra-to-hexahedra conversion and user-identified localized coarsening.

Chapter 7: Discussion

The GITTH application combined the TTH software with a mesh-management module designed to selectively coarsen regions within a hexahedral mesh. The coarsening module employed an arbitrary h-coarsening scheme to merge two sibling elements together into a single parent element. Since the main objective of the mesh-management module was to coarsen unstructured, non-orthogonal TTH-generated meshes, the coarsening algorithm had to contend with a few issues not regularly seen when coarsening good quality meshes. The degree of element distortion and the severity of irregular nodes within TTH-generated meshes created an environment that is not conducive towards h-coarsening. To accomplish localized coarsening, 8- to 26-node hexahedrons are used to efficiently grade the coarsened region and a number of validity checks are made to ensure this region does not break the rules outlined in Section 4.1. The validity checks comprise surface topology and element validity checks; such as edge and dimple checks for the surface topology and binary-grading and verification checks for element validity. To improve the quality of coarsened elements, a midedge node adjustment scheme is utilized to straighten all edges on the parent element.

TTH and the mesh-management module both use the TIAMAT format for mesh data files. The TIAMAT format has been used for two primary reasons. First, TIAMAT is able to handle the use of 8- to 26-node hexahedrons and second, there is existing software available to convert the mesh files to ones readable by LS-PrePost. If other solvers or pre-processors are required for simulations in the future, it is possible to implement subroutines to convert to their formats if required.

The GITTH application has demonstrated itself to be an effective interface for applying the TTH software to tetrahedral meshes and then coarsening select regions within a mesh. Evaluation of the coarsening algorithm has shown it to effectively coarsen orthogonal meshes and to a lesser extent, non-orthogonal TTH-generated meshes. Although the coarsening scheme is able to coarsen areas within poor quality meshes, this method does degrade the quality of the regions surrounding the coarsened elements of those irregular meshes.

As a minimum standard, the plate and cylinder cases were used to assess the effectiveness of the coarsening algorithm for good quality meshes. In both cases, the algorithm successfully coarsened elements with minimal distortion to any surrounding elements. The skew angle, aspect ratio and distortion parameter gave reliable quality measurements in most cases, but the skew angle did give misleading values for certain coarsened elements. In particular, the skew angle was found to give erroneous values for the coarsened elements from steps 8 and 12, shown in Table 6.3. These elements have a tapered shape but are not skewed to the degree that the skew angle calculations show.

For the coarse and fine Stationary Platen mesh cases, the coarsening algorithm was able to perform coarsening around most regions within the meshes. It was found in both cases that the midedge node adjustment during coarsening caused the quality of the surrounding elements to degrade somewhat. This degradation in quality limited the amount of coarsening in an area, since the distortion from consecutive coarsening attempts eventually leads to inverted or severely distorted elements.

It was found during testing that the initial mesh quality and severity of irregular nodes played a large role in the quality of coarsened elements and the distortion to

neighbouring elements. As mesh quality worsened, the neighbouring element distortion became greater. This could be seen when comparing the trends in the distortion parameter (DP) values between the fine (Figure 6.16) and coarse (Figure 6.12) mesh cases. Not unexpectedly, the fine mesh case showed a smaller degree of distortion to neighbouring elements.

The two main challenges for coarsening TTH-generated meshes, as discussed in Chapter 3, dealt with detecting and avoiding surface topology changes and severely distorted element creation. Surface topology subroutines such as the edge and dimple checks prevented any changes to the surface topology. The verification check implemented a methodology to prevent the creation of wedge-shaped elements. Since TTH-generated meshes were found to contain severely distorted elements, it was difficult to limit the creation of excessively distorted elements without greatly limiting the ability to coarsen such meshes. Therefore, the verification check was designed to reject the coarsening of elements that would cause wedge or inverted elements to form. Results from the study on wedge detection illustrated that the implementation of a $\pm 3^{\circ}$ angle tolerance between adjacent faces would provide reliable wedge detection.

In order for the coarsening software to demonstrate its effectiveness, it had to produce mesh data that would function within the TIAMAT FEA program. The successful run of a challenging, multiple step coarsened mesh through TIAMAT verified the coarsening algorithm produced valid mesh data. Although the coarsening scheme is able to coarsen elements within a TTH-generated mesh, further research is required to improve the degradation of surrounding elements and the overall quality of such meshes.

The obvious next step in this research path should include rigorous testing of TTH-generated and coarsened meshes.

Chapter 8: CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

1. The GITTH application provides a simple and straight forward approach for applying TTH to convert tetrahedral meshes to hexahedral meshes and then permitting local coarsening in user-selected regions.

   GITTH was written in C and developed for a Linux platform, using GTK+ and GNOME libraries.

2. The coarsening module working in conjunction with LS-PrePost offers a useful method for 2-element coarsening of hexahedral meshes in TIAMAT format using 8- to 26-node hexahedrons.

3. The coarsening scheme was developed to avoid two specific occurrences associated with TTH-generated meshes. First, topology checks ensure that the surface topology of the geometry does not change with coarsening. Second, a verification check ensures that coarsening does not create inverted or wedge-shaped elements.

4. The coarsening algorithm is able to create valid elements when coarsening both good quality, orthogonal meshes and poor quality, TTH-generated non-orthogonal meshes. Coarsening of TTH-generated hexahedral meshes causes deterioration to the quality of neighbouring elements, which may limit the amount of coarsening within a region.

8.2 Future Work

1. Studies of the initial level of distortion within TTH-generated meshes found that 15 to 20% of the elements within these meshes were severely distorted. The TTH software utilizes three smoothing schemes to help improve mesh quality, but the number of irregular nodes in the mesh limits these functions. A clean-up scheme [3] should be applied before the smoothing subroutines, to reduce the number of irregular nodes. This may help the performance of the smoothing subroutines.
2. Perform an optimization study to determine the optimal weight function (w) value for the 3-D isoparametric equation (3.1) used by TTH.
3. This thesis project has focused on 2-element h-coarsening of TTH-generated meshes. Coarsening non-orthogonal, irregular meshes leads to degradation in the quality of surrounding elements. A clean-up and smoothing scheme should be applied after coarsening to improve the quality of all elements within the surrounding region.
4. Coarsening algorithms for 4- and 8-element coarsening should be developed so that GITTH provides a useful range of coarsening functionality.
5. Both TTH-generated meshes and coarsened TTH meshes have not been subject to performance testing within a FEA solver. An assessment of the accuracy of such meshes is required. Therefore, a case study of these meshes should be performed to determine if these non-orthogonal meshes are suitable for regular use in FEA and whether they provide an improvement over tetrahedral meshes.

Chapter 9: REFERENCES

1 V. Adams and A. Askenazi, 'Building Better Products with Finite Element Analysis', Onward Press, Santa Fe, USA, pg. 1-26, 237-565, 1999.

2 F. Stasa, 'Applied Finite Element Analysis for Engineers', CBS publishing, New York, pg. 1-100, 1985.

3 S.J. Owen, 'A Survey of Unstructured Mesh Generation Technology', Proceedings of the 7th Int. Meshing Roundtable, Sandia National Lab, pg. 239-267, 1998. http://endo.sandia.gov/ (cited Jan. 2004)

4 S.E. Benzley, E. Perry, K. Merkley, B. Clark and G. Sjaardama, 'A Comparison of All Hexagonal and All Tetrahedral Finite Element Meshes for Elastic and Elasto-plastic Analysis', Proceedings of the 4th Int. Meshing Roundtable, Albuquerque, N.M., USA, 1995. http://endo.sandia.gov/ (cited Jan. 2004)

5 R.W. Leland, D.J. Melander, R.W. Meyers, S.A. Mitchell and T.J. Tautges, 'The Geode Algorithm: Combining Hex/Tet Plastering, Dicing and Transition Elements for Automatic, All-Hex Mesh Generation', Proceedings of the 7th Int. Meshing Roundtable, Sandia National Lab, 1998. http://endo.sandia.gov/ (cited Jan. 2004)

6 A. Carmona, 'Tetrahedra to Hexahedra Conversion for Finite Element Analysis', M.A.Sc. Thesis, Carleton University, Ottawa, Canada, 2002.

7 T.J.R. Hughes, K.S. Pister and R.L. Taylor, 'Implicit-Explicit Finite Elements in Nonlinear Transient Analysis', Comp. Methods in App. Mechanics and Engng., Vol. 17/18, pg. 159-182, 1979.

8 T. Rylander and A. Bondeson, 'Stability of Explicit-Implicit Hybrid Time-Stepping Schemes for Maxwell's Equations', Journal of Computational Physics, Vol. 179, pg. 426-438, 2002.

9 S.P. Wang, S. Choudhry and T.B. Wetheimer, 'Comparison between the static implicit and dynamic explicit methods for FEM simulation of sheet forming processes', MARC Analysis Research Corporation, Palo Alto, CA., USA. http://www.marc.com/support/Library/numiform1.pdf (cited Jan. 2004)

10 P.J. O'Rourke, D.C. Haworth and R. Ranganathan, 'Computational Fluid Dynamics', ASM Handbook Materials Selection and Design, Vol. 20, ASM International, Materials Park, OH, pg.186-203, 1997.

11 A. Filippone, 'Advanced Topics in Aerodynamics', Figure 4 – Airfoil grid(hyperbolic), Mech. Eng. Dept., Thermo and Fluids Division, Manchester. http://aerodyn.org/Frames/1cfd.html (cited Jan. 2004) Used with Permission from **www.aerodyn.org**.

12 J.M.J. McDill, Personal Communication, June, 2004

13 D.L. Dewhirst, Ford Motor Company, 'Finite Element Analysis', ASM Handbook Materials Selection and Design, Vol. 20, ASM International, Materials Park, OH, pg.176-185, 1997.

14 R. Cook, 'Concepts and Applications of Finite Element Analysis', 2nd edition, John Wiley and Sons, pg. 113-142, 1981.

15 J.M.J. McDill, J.A. Goldak, A.S. Oddy, and M.J. Bibby, 'Isoparametric Quadrilaterals and Hexahedrons for Mesh-Grading Algorithms', Communications in Applied Numerical Methods, Vol. 3, pg. 155-163, 1987.

16 J.M.J. McDill and A. S. Oddy, 'A Nonconforming Eight to 26-Node Hexahedron for Three-Dimensional Thermal-Elasto-Plastic Finite Element Analysis', Comp. & Structures, Vol. 54, No. 2, pg. 183-189, 1995

17 A. Kela, R. Perucchio and H.B. Voelcker, 'Toward Automatic Finite Element Analysis', Comp. in Mechanical Engineering, Vol. 5, No.1, pg. 57-71, 1986.

18 R. Schneiders, 'A Grid-Based Algorithm for the Generation of Hexahedral Element Meshes', Engng. with Comp., Vol. 12, pg. 168-177, 1996.

19 Y. Su, K.H. Lee, and A.S. Kumar, 'Automatic Hexahedral Mesh Generation for Multi-Domain composite Models Using a Hybrid Projective Grid-Based Method', Computer-Aided Design, Vol. 36, Issue 3, pg. 203-215, 2004.

20 K. Ho Le, 'Finite Element Mesh Generation Methods: A Review and Classification', Computer Aided Design, Vol. 20, pg. 27-38, 1988.

21 S.A. Coons, 'Surfaces for computer-aided design of space forms', Technical Report MAC-TR 44 MIT, Cambridge, MA, USA, 1967.

22 Sandia National Laboratories, Engineering Sciences, Research and Development. http://endo.sandia.gov/ (cited Jan. 2004)

23 M. Whiteley, D. White, S. Benzley and T. Blacker, 'Two and Three-Quarter Dimensional Mesh Facilitators', Engng. with Comp., Vol. 12, pg. 144-154, 1996.

24 Sandia National Laboratories, CUBIT 9.0, 'Mesh Generation Toolkit' http://sass1693.sandia.gov/cubit/help-version9.0/cubithelp.html (cited Jan. 2004)

25 T. Blacker, 'The Cooper Tool', Proceedings of the 5th international Meshing Roundtable, pg. 13-29, 1996.
http://endo.sandia.gov/ (cited Jan. 2004)

26 M. Lai, S. Benzley and D. White, 'Automated Hexahedral Mesh Generation by Generalized Multiple Source to Multiple Target Sweeping', Int. J. Numer. Meth. Engng., Vol. 49, pg. 261-275, 2000.

27 C.G. Armstrong, D.J. Robinson, R.M. McKeag, T.S. Li, S.J. Bridgett, R.J. Donaghy and C.A. McGleenan, 'Medials for Meshing and More', Proceedings of the 4th Int. Meshing Roundtable, Albuquerque, N.M., USA, 1995. http://endo.sandia.gov/ (cited Jan. 2004)

28 A. Sheffer, M. Etzion, A. Rappoport and M. Bercovier, 'Hexahedral Mesh Generation using the Embedded Voronoi Graph', Proceedings of the 7th Int. Meshing Roundtable, Sandia National Lab, 1998. http://endo.sandia.gov/ (cited Jan. 2004)

29 T. Blacker and M.B. Stephenson, 'Paving: A New Approach to Automated Quadrilateral Mesh Generation', Int. J. Numer. Meth. Engng., Vol. 32, pg. 811-847, 1991.

30 D.R. White, P. Kinney, 'Redesign of the Paving Algorithm: Robustness Enhancements through Element by Element Meshing', Proceedings, 6th International Meshing Roundtable, Sandia National Laboratories, pg.323-335, 1997. http://endo.sandia.gov/ (cited Jan. 2004)

31 T.J. Tautges, T. Blacker and S.A. Mitchel, 'The Whisker Weaving Algorithm: A Connectivity-Based Method for Constructing All-hexahedral Finite Element Meshes', Int. J. Numer. Meth. Engng., Vol. 39, pg. 3327-3349, 1996.

32 N.T. Folwell and S.A. Mitchell, 'Reliable Whisker Weaving via Curve Contraction', Proceedings of the 7th Int. Meshing Roundtable, Sandia National Lab, 1998. http://endo.sandia.gov/ (cited Jan. 2004)

33 S.J. Owen and S. Saigal, 'H-Morph: An Indirect Approach to Advancing Front Hex Meshing', Int. J. Numer. Meth. Engng., Vol. 49, pg. 289-312, 2000.

34 S.J. Owen, M.L. Staten, S.A. Canann and S. Saigal, 'Advancing Front Quadrilateral Meshing Using Triangle Transformations', Proceedings of the 7th Int. Meshing Roundtable, Sandia National Lab, 1998. http://endo.sandia.gov/ (cited Jan. 2004)

35 A. Oddy, J. Goldak, J.M.J. McDill, and M. Bibby, 'A Distortion Metric for Isoparametric Finite Elements', Transactions of the CSME, Vol. 12, No. 4, pg. 213-217, 1988.

36 P.M. Knupp, 'Achieving Finite Element Mesh Quality Vie Optimization of the Jacobian Matrix Norm and Associated Quantities: Part 1 – A Framework for the Surface Mesh Optimization', Int. J. Numer. Meth. Engng., Vol. 48, pg. 401-420, 2000.

37 P.M. Knupp, 'Achieving Finite Element Mesh Quality Vie Optimization of the Jacobian Matrix Norm and Associated Quantities: Part 2 – A Framework for volume mesh optimization and the condition number of the Jacobian matrix', Int. J. Numer. Meth. Engng., Vol. 48, pg. 1165-1185, 2000.

38 P.M. Knupp, 'Hexahedral and Tetrahedral Mesh Untangling', Engng. with Comp., Vol. 17, Issue 3, pg. 261-268, 2001

39 G.D. Kerlick and G.H. Klopfer, 'Assessing the Quality of Curvilinear Coordinate Meshes by Decomposing the Jacobian Matrix', Applied Mathematics and Computation, Vol. 10/11, pg. 787-807, 1982.

40 S.A. Canann, J.R. Tristano and M.L. Staten, 'An Approach to Combined Laplacian and Optimization-Based Smoothing for Triangular, Quadrilateral, and Quad-Dominant Meshes', Proceedings of the 7th Int. Meshing Roundtable, Sandia National Lab, 1998.
http://endo.sandia.gov/ (cited Jan. 2004)

41 L. Freitag, M. Jones and P. Plassmann, 'An Efficient Parallel Algorithm for Mesh Smoothing', Proceedings of the 4th Int. Meshing Roundtable, Albuquerque, N.M., USA, 1995.

42 'Genetic Algorithms', Principle of CAD/CAM/CAE Systems, Section 9.5, pg. 275-289, K. Lee, Addison-Wesley, 1999.

43 'Evolutionary Algorithms in Engineering and Computer Science', Edited by Kaisa Miettinen, Marko M. Makela, Pekka Neittaanmaki and Jacques Periaux, 'Chapter 1-Using Genetic Algorithms for Optimization: Technology Transfer in Action', John Wiley and Sons LTD., pg. 3-17, 1999.

44 W. Annicchiarico and M. Cerrolaza, 'Optimization of Finite Element Bidimensional Models: An Approach Based on Genetic Algorithms', Finite Elements in Analysis and Design, Vol. 29, pg. 231-257, 1998.

45 J. Sziveri, B. Cheng, A. Bahreininejad, J. Cai, G. Thierauf and B.H.V. Topping, 'Parallel Quadrilateral Subdomain Generation for Finite Element Analysis', Advances in Engineering Software, Vol. 30, pg. 809-823, 1999.

46 M. Holder and J. Richardson, 'Genetic Algorithms, Another Tool for Quad Mesh Optimization', Proceedings of the 7th Int. Meshing Roundtable, Sandia National Lab, 1998.
http://endo.sandia.gov/ (cited Jan. 2004)

47 A.Z.I. Salem, S.A. Canann and S. Saigal, 'Mid-Node Admissible Spaces for Quadratic Triangular 2-D Finite Elements with One Edge Curved', Int. J. Numer. Meth. Engng., Vol. 50, pg. 181-197, 2001

48 P. Kinney, 'Cleanup: Improving Quadrilateral Finite Element Meshes', Proceedings of the 6th Int. Meshing Roundtable, Sandia National Lab, 1997.
http://endo.sandia.gov/ (cited Jan. 2004)

49 B.A. Ammons and M. Vable, 'An HR-Method of Mesh Refinement for Boundary Element Method', Int. J. Numer. Meth. Engng., Vol. 43, Issue 6, pg. 979-996, 1998.

50 L.E. Lindgren, H.A. Haggblad, J.M.J. McDill, and A.S. Oddy, 'Automatic Remeshing for Three-Dimensional Finite Element Simulation of Welding', Comp. Methods Appl. Mech. Engng, Vol. 147, pg. 401-409, 1997.

51 H. Runnemalm and S. Hyun, 'Three-Dimensional Welding Analysis using an Adaptive Mesh Scheme', Computer Methods in Applied mechanics and Engineering, Vol. 189, pg. 515-523, 2000.

52 J.M.J. McDill and A.S. Oddy, 'Arbitrary Coarsening for Adaptive Mesh-Management in Three-Dimensional Automatic Finite Element Analysis', Math Modelling and Sci. Computing, Vol. 2, pg. 1072-1077, 1993.

53 J.M.J. McDill, A. Carmona, 'Tet-to-Hex Conversion for Finite Element Analysis', Proceedings of 8th Int. Conf. Numiform, Ohio, 2004.

54 L.R. Herrmann, 'Laplacian-Isoparametric Grid Generation Scheme', Journal of the Engineering Mechanics Division, Proceedings of the American Society of Civil Engineers, Vol. 102, pg. 749-756, October 1976.

55 J.M.J. McDill, A.S. Oddy, and J.A. Goldak, 'An Adaptive Mesh-Management Algorithm for Three-Dimensional Automatic Finite Element Analysis', Transactions of the CSME, Vol. 15, No.1, pg. 57-70, 1991.

56 R.E. Bank, A.H. Sherman and A. Weiser, 'Refinement Algorithms and Data Structures for Regular Mesh Refinement', Scientific Computing, pg. 3-17, 1983.

57 P.L. Baehmann, S.L. Wittchen, M.S. Shepard, K.R. Grice and M.A. Yerry, 'Robust, Geometrically Based, Automatic, Two-Dimensional Mesh Generation', Int. J. Numer. Meth. Engng., Vol. 24, pg.1043-1078, 1987.

58 G.F. Carey, M. Sharma and K.C. Wang, 'A Class of Data Structures for 2-D and 3-D Adaptive Mesh Refinement', Int. J. Numer. Meth. Engng., Vol. 26, pg. 2607-2622, 1988.

59 J.M.J. McDill, 'Package for Pre-Processing and Post-Processing of TIAMAT Data Files', Carleton University, Ottawa, 1996.

Appendix 1: GITTH USER MANUAL

Documentation and User Instructions For GITTH (GUI Interface to TTH software)

Carleton University
2004

Written by:

Version 2.0: (2004) Sandor Kostya

Version 1.0: (2002) Alejandra Carmona G. &
Andrew Rader

1 Introduction

Finite element analysis (FEA) is a popular numerical technique used in the analysis of engineering designs with complicated geometry and/or non-homogeneous material properties. This technique uses a computer model to approximate the mechanical behaviour of components under various environments and stress states. Finite element methods can be applied to two or three-dimensional models. Three-dimensional (3-D) models provide more accurate results, but at a cost of higher computational requirements. In order to apply FEA to a system, the object must be discretized into elements. In 3-D, these elements are generally tetrahedra or hexahedra.

Most commercial CAD/CAM packages such as Pro/ENGINEER or I-DEAS can be used for FEA mesh creation but are unable to fully mesh complicated geometries with hexahedral elements. Hexahedral elements are better suited for many FEA applications than tetrahedral elements. Because of this, it is desirable to have a computer program that facilitates the conversion of tetrahedral elements to hexahedral elements (TTH) to be used in FEA applications. The GITTH (GUI interface to TTH) application has been designed to meet this need. GITTH is comprised of an in-house TTH conversion tool [1] well as mesh management software which supports selective coarsening of hexahedral meshes.

This guide outlines the user instructions and documentation for the GITTH application.

2 Exporting tetrahedral meshes from commercial CAD packages: Pro/ENGINEER and I-DEAS

GITTH has been designed to work with tetrahedral meshes generated by Pro/Engineer and I-DEAS. Since the mesh file formats of both of these CAD packages are different, GITTH provides conversion functions which will alter both formats into one common format, in which the application TTH can read.

The steps below describe the methods for generating tetrahedral meshes from Pro/ENGINEER or I-DEAS models and exporting them as specific file types.

2.1 Pro/ENGINEER (2001 Release)

- First, start with a solid model open in the Pro/ENGINEER interface. From the toolbar, select **applications > mechanica** and check the **FEM mode** box.
- Second, from the toolbar, select **structure > model > mesh > create > solid > start**.
- The mesh size can be adjusted by selecting **create** or **edit** under **mesh control** on the menu on the right side of the screen.
- The mesh can then be exported by selecting **run**. ANSYS must be selected for exporting. This will give the Pro/ENGINEER file an ANSYS file extension (**.ans**). Enter a file name in the output filename field and save the file.

2.2 I-DEAS

- In the I-DEAS interface, create or open a solid model. From the menu at the top right of the screen, select **Simulation** mode instead of **Design** mode. Please note that it may take a few minutes to load. On the same menu toolbar, the **Master**

Model menu should now be a Meshing menu.

- Select the **Define Solid Mesh** (mesh picture at the top left square) from the top left square of the menu at the right side of the screen.

- In the **Create or Select FE Model** menu that appears, the model must be given a name. Then, select the volume to be meshed.

- The mesh can now be configured in the Define Mesh screen that appears. The mesh should be configured as desired by selecting the mesh type, element length, element family and element type. Select **Free** for the mesh type, **Solid** for the element family and **Linear** for the element type. The mesh can now be previewed selecting the **Preview Mesh** button at the bottom right corner. The mesh size and node numbering can be found by selecting the Modify Mesh Preview menu and pressing the **Estimate Solids** button. If the mesh is satisfactory, select **Keep Mesh**, otherwise the **Cancel Mesh** option should be selected to redefine the mesh properties.

- The mesh is exported by selecting **file > export** and selecting **I-DEAS Simulation Universal File (SUF)** for the file type.

3 Compiling GITTH

GITTH was written in C (programming language) on a Red Hat Linux platform. It was designed using the Glade user interface builder which is supported by GTK+ and GNOME libraries (both are Linux libraries written in C). Glade is a Linux GUI developer available in Red Hat and on the internet (http://glade.gnome.org/index.html).

3.1 Requirements

To compile and run GITTH, you require the following system/software:

- 800 MHz (recommended min.) or higher cpu speed,
- Red Hat 9.0 (or higher version) Linux,
- Glade 2.0 (or higher version) software installed,
- LS-PrePost Version 1.0 or higher (Only needed as a referencing application when using the coarsening tool within GITTH).

3.2 Compiling GITTH in Linux

- To compile GITTH, ensure the complete GITTH folder structure with files is transferred to a selected directory. All files and directories must have **user permissions** for read/write/execution (rwx). If any files/folders do not have user rwx permissions, GITTH may not compile properly.
- To gain rwx permission, a user must have root privilege or be the owner of the file. If either is true, type from within the shell: **chmod 777 *.** This will change all (owner/group/other) permissions to rwx.
- To compile, open a shell and change directories (**cd directory_name**) until you are within the GITTH main directory. Type: **autogen.sh.** This command uses the Glade 2 libraries to build a makefile. Change to the directory src (**ch src**). Type: **make.** The make command will compile the gitth source code. Once finished there should be a **gitth** file within the src directory. This is the execution file for GITTH.

4 GITTH User Instructions

GITTH has been designed to bring together the TTH application [1] and all of its peripheral software into a singular graphical interface. It is a user friendly interface providing access to the TTH software and to a selective hexahedral coarsening tool.

4.1 Directory Layout

The directory layout for GITTH is as follows:

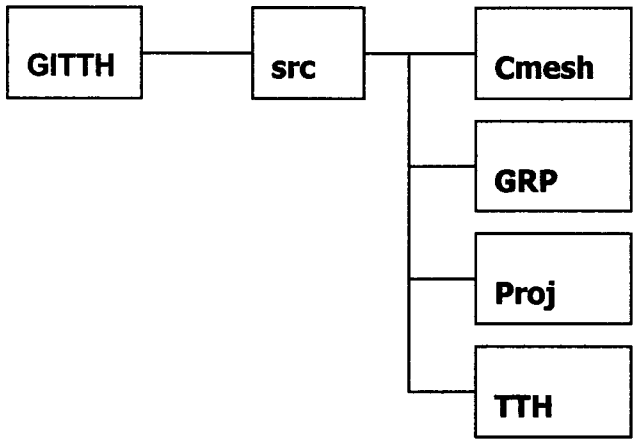


Figure 4.1: GITTH directory layout

The **GITTH** directory contains the Glade configuration and project files for GITTH. If any changes are required for the GITTH GUI, start the Glade software and open the file: **gitth.glade**. To open Glade from within Red Hat Linux, select the following menu selections: **Main Menu -> Programming -> Glade Interface Designer**. The Glade software will allow material to be added/removed as well as set up call functions for sub-routines. For changes to become effective, the current GUI design must be saved within Glade and then built through using the **Build** button on the Glade interface. Then follow the compile instructions from section 3.2 for the updated GUI to take effect.

The **src** directory contains the object and source files for the GITTH software. This includes the **main.c** source file as well as the interface and callback source files. These files are initially created by Glade. The **main.c** file and the **callbacks.c** source files

may be handled by a programmer. The other files are re-made by Glade during each compile, so they cannot be altered. If a Glade designed GUI has multiple working windows (i.e. message windows, file selection windows, etc.) that are not suppose to show during start-up, they must be removed from the **main.c** file. The **callbacks.c** file contains all the call functions for the GITTH interface. This file defines all GITTH global variables and contains all the button call functions for calling secondary windows or starting TTH/coarsening sub-routines.

The **Cmesh** directory contains all coarsening sub-routine files. These files have been specifically written using the GTK+ and Gnome libraries and will not compile without them.

The **GRP** directory contains the grpmsh.run executive file and its FORTRAN source code grpmsh.f. This program converts the mesh data into a binary form that LS-PrePost can open for viewing. This version of grpmsh has been programmed to work from within GITTH. It will not work efficiently as a stand alone program. Stand alone versions of this program are owned by Prof. M. McDill [2]. The grpmsh.run file has been compiled previously on Linux and should work on any machine running the same operating system (Red Hat Version 9.0 or greater). If it does not work from within GITTH, go to the **GRP** directory and type: **make –f Make_grpmsh**. This will recompile the grpmsh program.

The **Proj** directory is the defaulted folder for containing all project directories. GITTH will look for mesh data files within this directory, unless otherwise specified by a user.

The **TTH** directory contains all the TTH sub-routine source files. These files were

initially written to be a stand alone application but have since been converted to work with GITTH. As such, they were not designed for use within a GUI application and contains code that is memory demanding.

4.2 Starting GITTH

To start GITTH from the shell, change directories to src: "**cd ./GITTH/src**". Now type: **gitth**. This will bring up the main GITTH window. GITTH will post messages to the shell to inform the user of problems or when finished running certain sub-routines.

4.3 The GITTH Interface

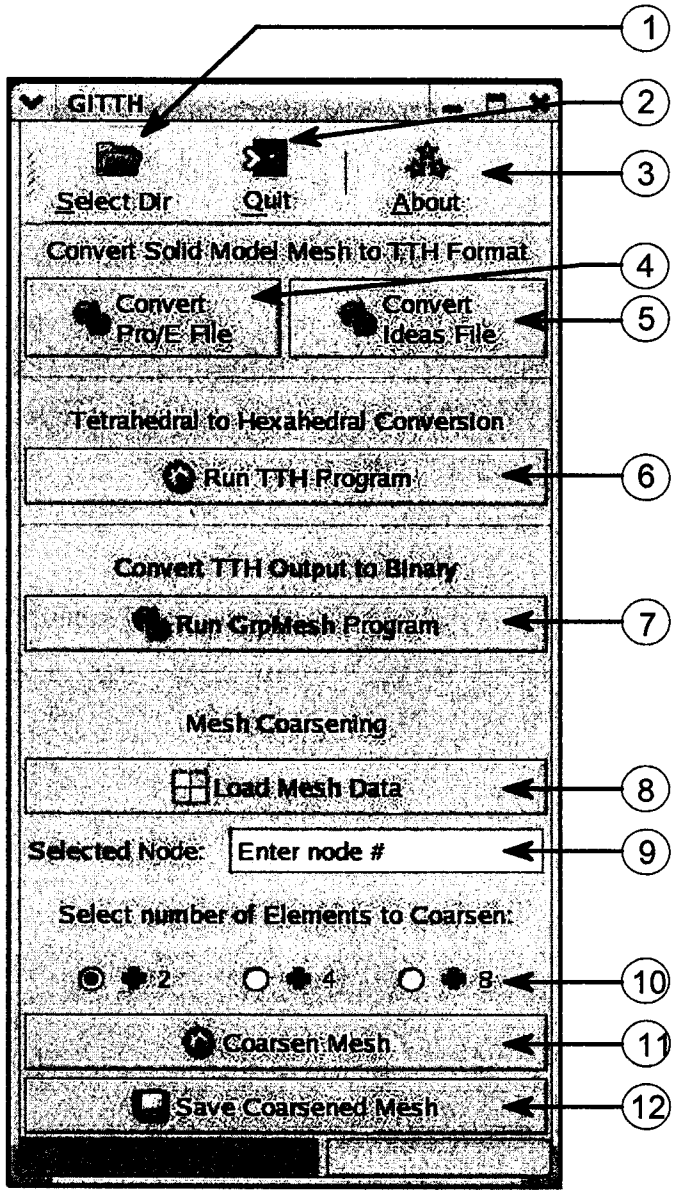


Figure 4.2: Main GITTH Interface

1) Select Directory Button: Selecting a project directory is necessary since all conversion software (Pro/E, Ideas and TTH) will try to open and/or save files to this selected folder. If a project directory is not selected, then the default directory **./GITTH/src/Proj** is used. For example, if TTH is run and it cannot find the file **output.txt** within the selected project directory, then it will return an error statement. This button opens a Project Selection window, asking for the directory in which the files for the current project are kept. The Project Selection window (Figure 4.3) contains a drop down list of the last 10 project directories used and a **Browse** button which opens a directory explorer window. Within the directory explorer window, a user may select or create a project directory. Hitting the **OK** button will select this folder, but does not close the window. The **Cancel** button must then be used to clear the explorer window. The selected folder will now appear within the Project Selection Window. Selecting **OK** will make this folder the current project directory.

Figure 4.3: Project Selection Window

2) Quit Button: Terminates GITTH and all its processes.

3) About Window: Opens a window containing information on GITTH and its authors.

4) Convert Pro/E File: This selection will open the Pro/E tetrahedral mesh data file (**.ans** extension) created in section 2.1 and convert it to TTH input file format. Selection of this button will open a file selection window, allowing a user to search for the specific

file. Once selected, the software will convert the mesh data format and save it to the file **output.txt** within the selected project directory.

5) Convert I-DEAS File: Like the previous button, this selection will open an I-DEAS tetrahedral mesh data file (**.unv**) created in section 2.2 and convert it to TTH input file format. The converted mesh data will be saved to the file **output.txt** within the selected project directory.

6) Run TTH Program: This button will run the TTH conversion software. It will look in the project directory for the file **output.txt**. If the **output.txt** file is not found, it will return an error statement. If the file is found, it will proceed to convert the tetrahedral mesh into an all hexahedral mesh. Once complete, it will save the new mesh data (in Tiamat format [3]) to the file **tth_output.txt** within the selected project directory.

7) Run GrpMesh Program: This button runs the Fortran program **grpmsh.run** to convert a hexahedral mesh file into a binary format in which LS-PrePost can open. Selection of this button will open a file selection window, allowing a user to select the specific mesh file for conversion. Once selected, the name of the file is saved to the file **./GITTH/src/GRP/grp_filesource.txt**. It then calls the **grpmsh.run** application. The **grpmsh.run** program looks for the filename in the **grp_filesource.txt** file and proceeds to convert that file. The new binary file will keep the same filename, except it now has "**grp_**" preceding the name and the extension becomes "**.bin**". For example, the **tth_ouput.txt** file becomes **grp_tth_output.bin**.

8) Load Mesh Data: This button loads mesh data into memory. Selection of this button opens a file selection window. A hexahedral mesh data file can then be selected for localized coarsening. If the file is not a mesh data file or it is not in Tiamat format, it will then reject the file selection.

9) Selected Node Window: Node id numbers are entered into this window for localized coarsening. The coarsening software uses the entered node number to coarsen neighbouring hexahedra surrounding this node.

10) Select number of Elements to Coarsen: The GITTH application hopes to give users the choice of selecting the number of elements to coarsen. This function gives preference for groups of 8, 4 or 2 elements to be coarsened. Presently, only the 2 element coarsening function is operating.

11) Coarsen mesh Button: Selection of this button starts the coarsening software. It grabs the node number entered in the Select Node window and passes it to the coarsening sub-routine. If the node number is invalid, it will output an error message.

12) Save Coarsened Mesh Button: Selection of this button will save the mesh data to file. A save file window (much the same as the File Selection window) will open, allowing a folder location and filename to be selected. This saved file can then be opened in GITTH using the Load Mesh Data Button at any later date.

4.4 Visualizing the Mesh with LS-PrePost

LS-PrePost is the pre-processing visualization software used by GITTH to visualize hexahedral meshes. LS-PrePost has been designed and developed by the Livermore Software Technology Corp (LSTC). The version used to date is: LS-PrePost Ver. 1.0.

LS-PrePost has various utilities to manipulate the mesh. One utility specifically identifies node identification numbers, so they can be input into GITTH for localized coarsening of the mesh. To open a file in LS-PrePost, select: **File> Open> Binary Plot**. Then select a binary file created using the GrpMesh button in GITTH. Once a mesh file is opened, select the **Ident** button on the top right hand side menu. A second menu will appear in

the bottom right hand side of the screen. The node setting should be the default setting. Now click on node locations within the window containing the mesh image. The selected node number will be displayed on screen. These node numbers can be input into GITTH when using its coarsening application. The coarsening software will coarsen the elements surrounding that node. To see the newly coarsened mesh, save the mesh data (**Save Coarsened Mesh button**[1]), convert the file to binary (**Run GrpMesh Program button**[2]) and reopen it in LS-PrePost.

4.5 GITTH Instructions

4.5.1 TTH Instructions

For converting a tetrahedral mesh into a hexahedral mesh from within GITTH, see the following steps:

Step 1. Select or create a project folder using the **Select Dir** Button (Button 1).

Step 2. Convert the tetrahedral mesh file to the required TTH input format by selecting either **Convert Pro/E or I-DEAS** (Buttons 4 or 5) depending on the file type.

Step 3. Select **Run TTH Program** (Button 6). Depending on the size of the mesh, it may take some time to convert the mesh.

Step 4. To view hexahedral mesh, select **Run GrpMesh Program** (Button 7). Select **tth_output.txt file** in project folder. Open LS-PrePost and follow instructions for opening binary file in section 4.4.

4.5.2 Coarsening Instructions

For selective coarsening of a hexahedral mesh from within GITTH, follow the steps below:

[1] Button 12 from section 4.3 - The GITTH Interface
[2] Button 7 from section 4.3 - The GITTH Interface

Step 1. Select **Load Mesh Data** (Button 8) to load mesh data into memory. If the **TTH conversion program** (Button 6) has recently been run, that particular mesh data will already be in memory.

Step 2. Open LS-PrePost and follow the instructions for opening a hexahedral mesh file in section 4.4. Using LS-PrePost, manipulate the mesh as required to find areas for coarsening. Identify node numbers in these areas and input them into GITTH (Window 9).

Step 3. There are a few rules the coarsening algorithm follows to ensure that the model topography or element shapes are not adversely affected:

1) Elements along an edge (internal or external) can not be coarsened.

2) Elements on the surface with a greater than 20° change of their surface normal compared to neighbouring element surfaces cannot be coarsened.

3) Coarsening that creates wedge shaped elements or holes in the mesh, are rejected.

Step 4. To coarsen, select **Coarsen Mesh** (Button 11). The software will give notice if coarsening was successful or unsuccessful. Continue to add node id's and coarsen as required.

Step 5. Select **Save Coarsened Mesh** (Button 12) to save mesh data to file. Use **Run GrpMesh Program** (Button 7) to convert data to binary for viewing coarsened mesh in LS-PrePost.

5 REFERENCES

[1] Alejandra Carmona Garcia, 'Tetrahedra to Hexahedra Conversion for Finite Element Analysis', M. A. Sc. Thesis, Carleton University, Ottawa, Canada, 2002.

[2] McDill, M.J., 'Package for Pre-Processing and Post-Processing for Tiamat', Carleton University, Ottawa, 1996.

[3] Oddy and McDill, 'Documentation for Tiamat', OMNIS, Ottawa, 1999.

Appendix A. LIST OF FILES AND FUNCTIONS

Table A-1: GITTH GUI Source Files (Location: ./GITTH/src)

main.c	**callbacks.c**
main	on_main_window_delete_event
	on_project_dir_button_clicked
support.c	on_quite_button_clicked
(do not edit this file – it is generated by Glade)	on_about_button_clicked
lookup_widget	on_proe_button_clicked
create_pixmap	on_ideas_button_clicked
create_pixbuf	on_tth_button_clicked
glade_set_atk_action_description	on_ps_okbutton_clicked
	on_ps_cancelbutton_clicked
interface.c	on_fs_ok_button_clicked
(do not edit this file – it is generated by Glade)	on_fs_cancel_button_clicked
create_main_window	on_done_okbutton_clicked
create_about_window	on_grpmesh_button_clicked
create_file_selection	on_error_okbutton_clicked
create_done_window	on_load_button_clicked
create_project_selection	on_save_cmesh_clicked
create_error_window	on_sm_cancel_button_clicked
create_cmesh_filesave	on_sm_ok_button_clicked

Table A-2: Coarsening Source Files (Location: ./GITTH/src/Cmesh)

gitth_coarsen.c	**Binary_Conversion.c**
coarsen_mesh	grpmsh_conv
coarsen2	store_filename
NLrelations.c	**Calc_AR.c**
sibling_search	aspect_ratio
element_counter	shape_function
nodeId_index	Jacobian
node_position	node_points
face_match	faceAngle_check
EdgeNode_2eCheck	
naybor_elements	**ext_surface.c**
CornerNode_2eSelection	edge_check
EdgeNode_2eSelection	free_surface
FaceNode_2eSelection	normal3d
return_backupData	elementEdge_check
node_shift_check	Lsurf_edgeCheck
	surfacePucker_check
tth_dataload.c	
tth_loadData	**initialize.c**
cmesh_saveData	array2D_init0
	array2D_init0f
	array1D_init0
	array1D_init0f

Table A-3: TTH Source Files (Location: ./GITTH/src/TTH/)

tth.c tth **4X4det.c** ThreeXThree FourXFour VolumeTetra **AR.h** CALCELXYZ CALCDPDR Vectorprod AR SKEWWELEMENT SKEWS NEWSKEWS BadskewsElementDisplay **AR_TET.h** ARTET GrphSkews SFHTET HexLocal2GlobalCoord SkewRotate **GRPHARHEX** SRcoormove **ARFix.h** ARFixCases ARFix **ARTet2HexStat.h** ARTet2HexStat **c_conect.h** CONECT SMOOTHNODE NDPLACE NDNAYELEM **c_JACOB3D.h** JACOB3D INVJ	**c_SFHBR.h** SFHBR **extra.h** RepeatNdDetector ElPerNDDisplay NumberOfBadElements onehex **findrst_c.c** POINTS3D_C FINDRST_C **fulgar_c.c** FACECHK FULGAR GrphARTET.h GrphARTET **pt2.h** (CellTable; Pttables explanations and limits) **Pt22.h** : PTGet_ ptnew_ ptput_ Functions contained in the file but not in use: PTCheck, PTMakeKey, PTEqual, PTEqualPoint, PTLessPoint, PTGreaterPoint, PTLessEqualPoint, PTIsIn, PTIntPut, PTInq_ PTGetR, PTGetC, PTSave, ptrest_ **Tetcentroid.c** TETCENTROID **isolap.c** ISOLAP **gpvalues.c** GPVALUES

Appendix B. FUNCTION DESCRIPTION

AR
Finds the aspect ratios of the element using Kerlick and Klopfer equations.

ARFix
Smoothing function to repair the hexahedra aspect ratios.

array2D_init0
Initializes a 2D integer array to zero.

array2D_init0f
Initializes a 2D double floating point array to zero.

array1D_init0
Initializes a 1D integer array to zero.

array1D_init0f
Initializes a 1D double floating point array to zero.

ARTET
Calculates the aspect ratio of tetrahedra.

aspect_ratio
Calculates the det J (at Gauss points, nodal points and centroid), aspect ratio and skew angle for an element.

BadskewsElementDisplay
Creates a file listing the extreme skew angle of each element and determining if the value is out of range.

CALCDPDR
Calculates the shape functions derivatives of (r, s, t).

CALCELXYZ
Translates the nodal coordinates of an element into specified arrays.

cmesh_saveData
Saves mesh data (element and nodal information) from memory to a file.

coarsen_mesh
Initiates the coarsening process once the coarsen button is selected on GITTH. It grabs the node number and number of elements to coarsen (2, 4, or 8 - coarsen_case) from the GITTH interface. It then calls the coarsen_case subroutine. Once a set of element

have been merged, it removes any redundant nodes and reorders the element identification numbers.

coarsen2
Coarsens 2 elements into a singular parent element.

CONECT
Subroutine used to find the connecting nodes from neighbour elements.

CornerNode_2eSelection
Locally orders the corner nodes of the new coarsened element.

EdgeNode_2eCheck
Checks if either of the 2 elements for coarsening already have edge nodes. If either has previous edge nodes along the new element edge, it then return a value of (0).

EdgeNode_2eSelection
Locally orders the edge nodes of the new coarsened element. The edge nodes are the 4 matching corner nodes of the 2 elements to be coarsened.

edge_check
Checks if passed node is on an edge of the model.

element_counter
Returns the number of elements that share a specified node.

elementEdge_check
Checks if an element is situated at the edge of a model.

ElPerNDDisplay
Displays the nodes numbers and coordinates of an element.

faceAngle_check
Calculates the dot products from the normal vectors of each face of an element. A value of (0) will be returned if edge sharing faces have an angle difference of 0 (+- a tolerance value)

FACECHK
Checks all 6 faces of a 3-D element to see if it is connected to other elements.

face_match
Checks if the 4 corner nodes of an element match the 4 corner nodes of a neighbouring element.

FaceNode_2eSelection
Locally orders the face nodes of the new coarsened element.

findrst_c
Given a point in space and the element it is within, this function finds the local (r, s, t) coordinate within the element.

FourXFour
Calculates the determinant of a 4x4 matrix.

free_surface
Finds surface element faces and nodes of entire mesh (used by GITTH).

fulgar
Finds external faces in the 3-D mesh (used by TTH).

GraphARHex
Classifies in intervals the values of the hexahedra aspect ratios. The values can be exported to spreadsheet programs for further analysis.

GraphARTET
Classifies in intervals the values of the tetrahedra aspect ratios. The values can be exported to spreadsheet programs for further analysis.

GrphSkews
Classifies in intervals the values of the hexahedra skew angles. The values can be exported to spreadsheet programs for further analysis.

grpmsh_conv
Calls the "grpmsh.run" FORTRAN application to convert a hexahedral mesh file to binary.

INVJ
Calculates the inverse of the Jacobian matrix in a hexahedral element.

Jacobian
Calculates the Jacobian matrix of a hexahedral element (for GITTH application).

JACOB3D
Calculates the Jacobian matrix of a hexahedral element (for TTH application).

Lsurf_edgeCheck
Checks if the free surfaces of two neighbouring elements share a node.

naybor_elements
Finds all elements that are neighbouring the elements to be coarsened.

NDNAYELEM
Returns the number of neighbouring elements for the specific node.

NDPLACE
This procedure selects the two nodes to smooth. This selection is based on the skew of the element and the position i.e. interior, exterior node. The coupling nodes should be placed in opposite faces to each other and at least one of them should be an interior node.

NEWSKEWS
Calculates the skew angles in an element.

nodeId_index
Returns the nodeData[][] array index number for the passed node id.

node_points
Returns an array of either gauss points or nodal point (r, s, t) coordinates.

node_position
Find the local position of a node within an element. Returns a value of (1) for a corner node, (2) for an edge node and (3) for a face node.

node_shift_check
Checks if a node has shifted position, away from its original location.

normal3d
Finds normal vector of an elements face.

NumberOfBadElements
Creates a file that displays the number of elements in the mesh and the number of badly shaped elements. It also gives the number of elements with poor aspect ratio, low or high skew angles. It also displays the global extremes for both aspect ratios and skew angles.

onehex
Creates a file to visualize a hexahedral element in Pro/ENGINEER.

POINTS3D_C
Transforms global coordinates to local coordinates.

Pt_get
Looks for information in K-D tree data structure.

Pt_new
Creates a new record in K-D tree.

Pt_put
Put new record in K-D tree.

Pt_table
Creates a K-D tree.

RepeatNdDetector
Detects duplicate nodes.

return_backupData
Returns the original element and nodal coordinate data (of elements to be coarsened) to their elementData and nodeData arrays if the new element is rejected.

SFHBR
Determines the value of the shape functions for a hexahedral element.

SFHTET
Determines the value of the shape functions for a tetrahedral element.

shape_function
Calculates the shape functions and derivatives for an element for a set of (r, s, t) coordinates.

Sibling_search
Finds element-node relations and stores it within the array NAYBORS[][].

SkewRotate
Smoothing function that repairs the skew angles of distorted hexahedra.

SKEWS
Calculates the skew angles in an element. It stores the out of range element numbers and angles.

SKEWSELEMENT
Creates a file with the skewed elements values and their nodes coordinates.

SMOOTHNODE
Isoparametric smoothing function.

SRcoormove
Prints in a file the coordinates of a specific element during the smoothing process.

store_filename
Stores the selected filename for grpmsh conversion in the file:
"./GITTH/src/GRP/grp_filesource".

surfacePucker_check
Checks if the merging of two elements will cause the mesh surface to pucker.

Tetcentroid
Computes the new value of the interior node for distorted tetrahedral elements.

ThreeXThree
Calculates the determinant of a 3x3 matrix.

tth_loadData
Loads the hexahedral mesh data to memory from a file.

Vectorprod
Determines the dot product between two vectors.

VolumeTetra
Calculates the volume of a tetrahedral element.

Appendix 2: NORMAL VECTOR COMPARISON

The normal vector of a surface is calculated using the cross product of the two vectors, **V13** and **V24**. Figure A2.1 illustrates this method.

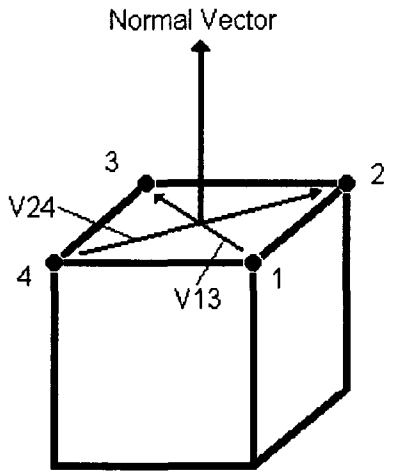


Figure A2.1: Normal Vector Calculation

To determine **V13** and **V24**, the following equations were used:

$$\begin{aligned} \mathbf{V13}(X) &= X3 - X1 & \mathbf{V24}(X) &= X4 - X1 \\ \mathbf{V13}(Y) &= Y3 - Y1 & \mathbf{V24}(Y) &= Y4 - Y1 \\ \mathbf{V13}(Z) &= Z3 - Z1 & \mathbf{V24}(Z) &= Z4 - Z1 \end{aligned} \qquad \text{(A2-1)}$$

where; X, Y, Z are the global coordinates and

1, 2, 3, 4 are the local node numbers.

The normal vector (**n**) is found by calculating the cross product of vectors **V13** and **V14** as shown:

$$\mathbf{n} = (\mathbf{V13}) \times (\mathbf{V24}) \qquad \text{(A2-2)}$$

This can be broken down into the x, y, z coordinates:

$$\begin{aligned} \mathbf{n}(x) &= \mathbf{V13}(Y) \times \mathbf{V24}(Z) - \mathbf{V13}(Z) \times \mathbf{V24}(Y) \\ \mathbf{n}(y) &= \mathbf{V13}(Z) \times \mathbf{V24}(X) - \mathbf{V13}(X) \times \mathbf{V24}(Z) \\ \mathbf{n}(Z) &= \mathbf{V13}(X) \times \mathbf{V24}(Y) - \mathbf{V13}(Y) \times \mathbf{V24}(X) \end{aligned} \quad \text{(A2-3)}$$

The unit normal vectors are used for comparing vector directions. To determine the unit normal vector (**N**), the normal vector (**n**) is divided by its magnitude (mag).

$$\text{mag} = [\, \mathbf{n}(X)^2 + \mathbf{n}(Y)^2 + \mathbf{n}(Z)^2]^{1/2} \quad \text{(A2-4)}$$

$$\begin{aligned} \mathbf{N}(X) &= \mathbf{n}(X) / \text{mag} \\ \mathbf{N}(Y) &= \mathbf{n}(Y) / \text{mag} \\ \mathbf{N}(Z) &= \mathbf{n}(Z) / \text{mag} \end{aligned} \quad \text{(A2-5)}$$

There are two cases within the coarsening algorithm where the unit normal vectors are compared. The first case, shown in Figure A2.2 (a), compares neighbouring surface unit normal vectors to determine if any node or edge is on an edge. The second case, shown in Figure A2.2 (b), compares neighbouring face surface unit normal vectors to determine if an element has become wedge-shaped.

The unit vector directions are compared by finding their dot product (dotp).

$$\text{dotp} = \cos^{-1} [(N_A(X) \cdot N_B(X)) + (N_A(Y) \cdot N_B(Y)) + (N_A(Z) \cdot N_B(Z))] \quad \text{(A2-6)}$$

where; $_{A,B}$ are the two unit normal vectors and

dotp is the angle difference between unit vectors.

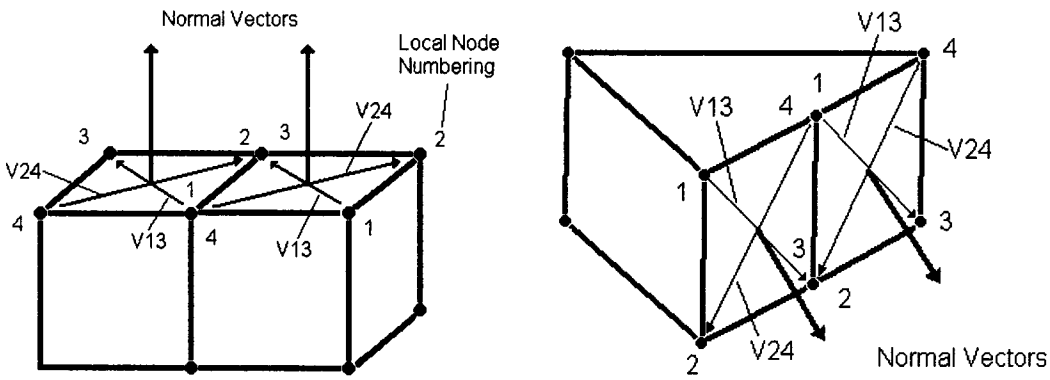


Figure A2.2: Comparison of normal vectors. (a) Edge check (b) Wedge check

For edge checks, if the angle between the two unit normal vectors is greater than 20°, then the two free surfaces are considered to reside on an edge. For the wedge check, if the angle between unit normal vectors is 0° ± 3°, then the element is said to be wedge-shaped.