Ship Science Report 109
Efficient multi-level adaption methods for unstructured polyhedral computational meshes

A. M. Wright
S. R. Turnock
School of Engineering Sciences, University of Southampton

May 1999

Abstract

The definition of a computational mesh uniquely suited to a given model is a difficult and time consuming process. Using spatial and flow property quality definitions a method has been developed to adapt and optimise any given mesh. Through complex data structures, efficient surface definition and statistical analysis efficient cellular and interface manipulation routines are initiated. Cellular fission/fusion, nodal convection and interface manipulation are detailed and benefits of the adaption listed. The development of such techniques into a fully automated grid generation process is discussed.

Contents

List of Figures

1 Introduction

1.1 Importance of mesh adaption

Due to the discrete modelling of physical continuum by computational methods, there is inevitably some dependence of the solution upon the discrete positioning of data points used.

The general trend in large scale CFD problems is to use fine grids as much as possible to resolve physical flow features. Mesh independence is sought via using finer and finer meshes until the solution no longer changes to any significant amount. With complex three dimensional flow this can result in massive problems, both in memory storage requirements and CPU time. This relatively crude method of gaining mesh independence is due to the fact that meshes are arbitrarily generated, with only the operators knowledge and experience dictating where points should be concentrated. This method may also have undetected, underlying geometric pathologies such as cellular twist that will introduce secondary mesh dependencies, just as strong as those being removed [1].

Thus it can be reasonably concluded that the user should be removed from the mesh definition; they should instead be carried out via a form of automated adaptive process. Recent years have seen a rapid development of such processes, normally based on some form of a posteriori error [2, 3, 4].

The major advantage of such methods over the arbitrary use of finer meshes is the saving in resources, principally computational time and memory. With the adaption of the computational mesh via quantifiably selected criteria, only the areas requiring refinement are altered. Thus the number of geometrical entities is minimised for a given solution accuracy, resulting in lower memory storage requirements and lower CPU usage in the solution phase. In summary, mesh adaptation increases solution efficiency for a given level of accuracy.

A second reason for the use of mesh adaptation, one that is often overlooked, is in areas on complex geometry. Boundaries with large curvature in close proximity of each other often lead to very poor geometrical grid quality. Such a distribution of mesh points often causes numerical instabilities in the solution. Adaption of the mesh, to better suit the point distribution to the given geometry, reduces such instabilities and creates a better behaved solution.

1.2 Methods and control of adaption

The two main methods of grid adaptation in use are stretching and splitting, or variations of either. The easiest method, stretching, relocates mesh points, clustering them in areas of high flow gradients. Such methods have been used successfully for a number of years [5, 6, 7], most often for supersonic applications where accurate modelling of the shock wave is required. Advantages of this method are the lack of creation or destruction of geometrical entities while a common problem is excessive mesh movement, resulting in highly skewed grids.

Cellular splitting to gain greater resolution is more computationally expensive than relocation of existing entities but avoids creating skew in the mesh. By avoiding such geometrical issues, accuracy (particularly in viscous applications) is improved and dissipative errors are reduced [8].

The instigation, and subsequent control, of mesh adaptation is normally achieved by the use of a flow property gradient weighting function. Control volumes are concentrated in areas of high flow gradients, and removed from areas of low gradients. More complicated and generalised schemes have been divised, such as that by Brackbill and Saltzman [9] which incorporates grid smoothness, orthogonality and volume variation as well as the basic gradient weighting function.

The work presented in this report uses a mixture of methods to instigate the mesh adaptation, both geometrical and flow solution based. The actual method of adaption also varies, and is chosen according to the manner in which the geometrical entity was selected. If flow gradients are too large control volumes will be split, where as if a face has low planarity, its vertices will be moved to compensate. Thus the method of adaption will be that best suited to the local problem.

Chapter 2 details previous work in the topic, highlighting problems encountered and advantages of the various techniques. Chapters 3 and 4 detail the methods used to control the adaption process and the underlying data structures that allow the process to occur. Chapters 5 to 7 detail the algorithms used to achieve the adaption process. Finally Chapter 8 concludes the work achieved and lists a number of possible future developments.

2 Literary review

It is now commonly recognised that grid adaptation is a manner in which solution accuracy can be improved [10, 11], without recourse to higher order numerical schemes. Other uses of grid adaptation are to increase efficiency through mesh optimisation; maintaining the same overall number of control volumes, but re-distributing them to obtain the best possible solution [1]. The third possible use of grid adaptation is within a multigrid scheme, where recursive sub-division of cells allows the different grids to be created.

2.1 Control of adaption

There are three main ways in which a mesh can be adapted; an error estimator function, geometric limitors or flow solution limitors.

Geometrical limitors have probably been in use for the greatest length of time in some form. The general techniques used in unstructured mesh generation of advancing front and Delanay triangulation are geometrically controlled mesh adaptation [12, 13], Since then more refined techniques have been utilised to increase mesh quality, incorporating edge lengths, smoothness and domain boundary curvature [7, 14, 15, 16, 17]. These methods are still relatively basic, being dependant upon mesh element size in the main. Further developments, incorporating mesh orthogonality, are now being promoted by Jacquotte, McRae and Laflin, and Hassan and Probert [18, Chapters 33-35], and the relative importance of cell alignment and skew have been commented upon [18, Section 34.6].

The second method of controlling mesh adaption is error estimation. This is where an estimate is obtained by the difference between the computed solution, constant by element, and a reconstructed linear solution in order to provide a measure of the grid quality; that is, the ability to correctly represent the flow characteristics. The concept underlying this process is to have the error constant throughout the domain, thus minimising the global error of the solution [1]. This method has been used successfully by a number of parties [3, 4], however the major drawback of these methods is that they do not correctly take account of the error due to non-uniform grid spacing [11]. Another problem of this technique is the tendency to produce isotropic meshes due to the underlying principle of making the length scale of the elements the same in all directions. This method works satisfactorily for flows possessing large gradients in all directions, but not for highly directional flow features such as wakes and vortices [1]. Habashi et al [1] have more recently devised an error estimator that eliminates most of these problems, but as yet only in two dimensions.

By far the most common method of controlling mesh adaption is the assessment of local flow properties with respect to some predefined limit. Error estimator often use flow properties to define the error, but not as the primary qualifier. Examples of this technique most commonly use the local Mach number or density for compressible flows [19] and pressure or velocity for low Mach number or incompressible flows [20, 21, 22, 23]. Parthasarathy and Kallinderis [20, 21] utilise a cell tree data structure to increase the efficiency of the solver, and allow both cell division and cell merging to allow increased accuracy without increasing the problem size greatly. The advantage of using the local flow gradients to control adaption is that it's an easy to apply and proven technique, however it can cause the grid to become highly non-uniform and contain areas of poor geometrical grid quality.

2.2 Method of adaption

There are two key methods in use presently in order to optimise a given mesh; either refinement of the mesh or redistribution of the mesh. Changing the solution method to a higher order scheme has been used, but has no significant application in field solvers for multi-dimensional problems as yet [18, Part IV].

Mesh refinement requires the addition (or removal) of data points locally in regions of relatively large (or small respectively) error. Problems associated with this method are increased computer time and storage as well as difficulties with the data structure. Unstructured meshes are particularly suited to such a method [23]. Tree data structures remove many of the data handling problems [4, 21], and the basic algorithms used during this operation lend themselves very easily to creating a multigrid solution to further increase the efficiency of the solution. Cartesian meshes have also been used successfully for mesh refinement [15], but "the method relies on being able to cope with exceptions and is therefore more verbose". In addition, only two dimensional problems have been presented, which suggests that the level of complexity of the algorithms required for realistic shapes will be much greater.

The second method of adapting a mesh is the redistribution of the data points. In this approach points move from areas of relatively low error to areas of large error, thus aiming for an equal distribution of error throughout the domain. The most obvious advantages of this system are the ability to maintain the same mesh connectivity and the lack of requirement to create and destroy entities. It is also inherently possible to use structured grids, thereby utilising the efficiency of such methods. Structured mesh adaption has been employed [6, 7, 14, 19] with good results. An added benefit of this approach is that the grid speed can be mathematically defined simultaneously with the grid movement, thus increasing CPU efficiency [19].

From the study of previous publications it has been concluded that both flow and geometrical quality must be assessed to correctly assess grid quality. A further conclusion is that mesh refinement and redistribution should be used simultaneously to optimise the mesh.

3 Grid quality definition

Qualitative and quantitative measurement of the confidence level with which the results of computational fluid dynamics (CFD) codes can be used has received increasing levels of attention as numerical simulations become more commonly used and depended upon within the design environment [24][25]. For a numerical simulation to be used effectively and efficiently there must be a high degree of trust in the quality of a given solution. Sources of error and uncertainty are numerous, but one of the most commonly identified is that of the geometrical mesh.

To clarify the problem the definition of quality must first of all be stated. "Quality" is "a distinctive attribute or faculty" or "the relative nature or kind or character of a thing", while "quality control" is a system for maintaining standards ... against the specification" [26]. In this case the object is the discrete mesh, and the acceptable levels are the accuracy of the final flow solution and the resources required (be they time, computing facilities or technical ability) to gain that solution. Thus a linking of the grids physical properties and the flows properties must be created. From this a direct relationship between the grids physical characteristics and the flows accuracy can be derived.

The spatial accuracy of a numerical solution is commonly termed the discretisation error. Strictly speaking this term only covers the local truncation error from the use of discrete piece-wise constant entities in the place of a continuous domain. In two and three dimensional problems there is another possibility for error, closely linked to the discretisation error of the numerical model, and often incorporated within the generic grouping of grid errors. For this work such geometrical deviations from orthogonal and equally spaced control volumes has been termed the *distortion* error.

Work carried out by Wright and Turnock [27] has assessed local numerical error of a solution with respect to the local geometrical dimensions in terms of both discretisation and distortion error. Underlying discretisation error can be assessed and removed by a grid dependency study, as suggested by Eça and Hoekstra [28],leaving the distorting effect of the various geometrical properties. Polynomial curve fitting or least squares error fitting can then be applied to link generic geometric trends to the error estimation. This method has been applied to the flow solver described by Wright and Turnock [29].

3.1 Error Analysis of Numerical Solution

In a one dimensional problem there are three distinct levels of error between the physical reality and the numerical solution; the neglecting of physical parameters; the error incurred in the representation of the physical world by a mathematical equation; and the error of representing a continuous domain by a discrete numerical mesh [30]. The order of accuracy of any scheme, both temporally and spatially, can be defined from truncation error analysis, and the stability via Von Neumann analysis or a similar method. This provides a discretisation error in terms of an order of Δx and Δt. Figure 1 graphically illustrates this breakdown of the error into modelling assumptions and mis-representations, discretisation error and the distortion error. As the grid becomes coarser the discretisation error increases and so also can the distortion error. It must be noted that as with coarser meshes the discretisation error *will* increase, but the distortion error may not, depending as it does upon properties that are non-dimensional, and hence not dependant upon cellular size. The current work deals only with a first order Euler code, but this should work as an advantage, highlighting the effects of grid distortion that can cause problems in higher order viscous solvers.

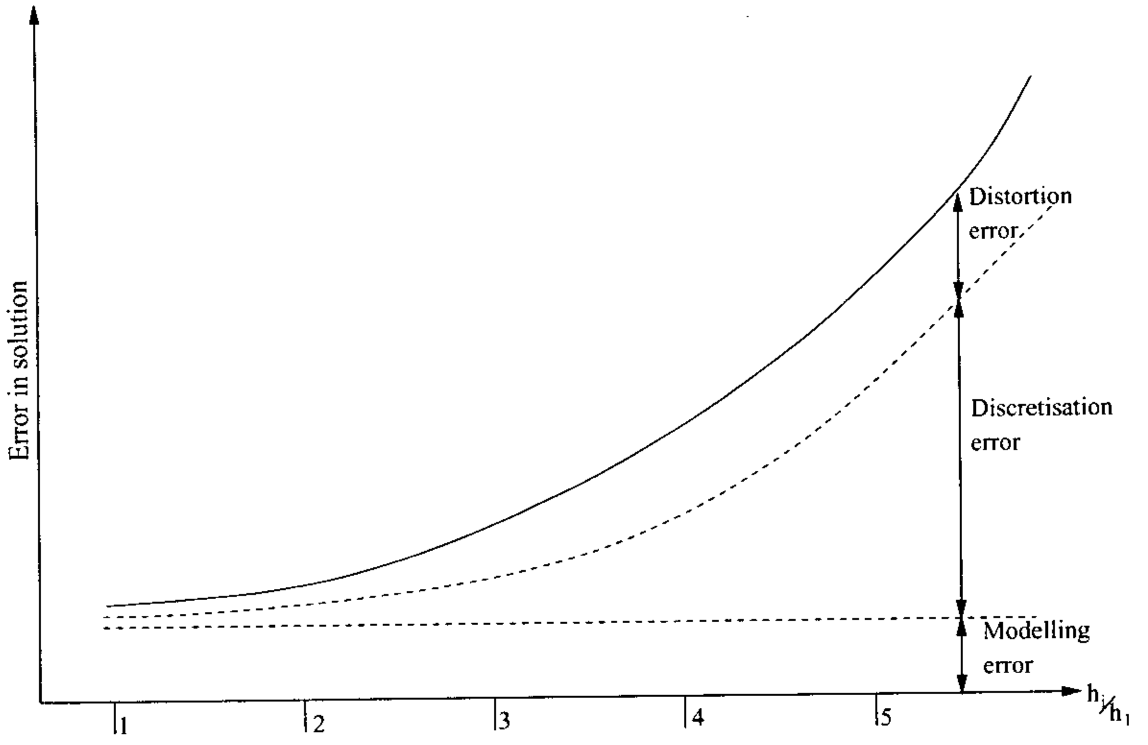


Figure 1: Division of error for mesh size variation

The dependence of a solution upon the discrete mesh has been further observed and quantified, via forms of the Richardson extrapolation, as well as general discussion of the definitions of error and verification, by Stern et al [31] and Eça and Hoekstra [28].As noted at the beginning of this Chapter, as well as this error due to the use of discrete entities, the lack of orthogonality and planarity within the mesh inserts another source of error. The majority of research concerning the topic of grid quality has tended to be of a qualitative nature, rather than a quantitative nature, and in total the topic has been given little theoretical definition.

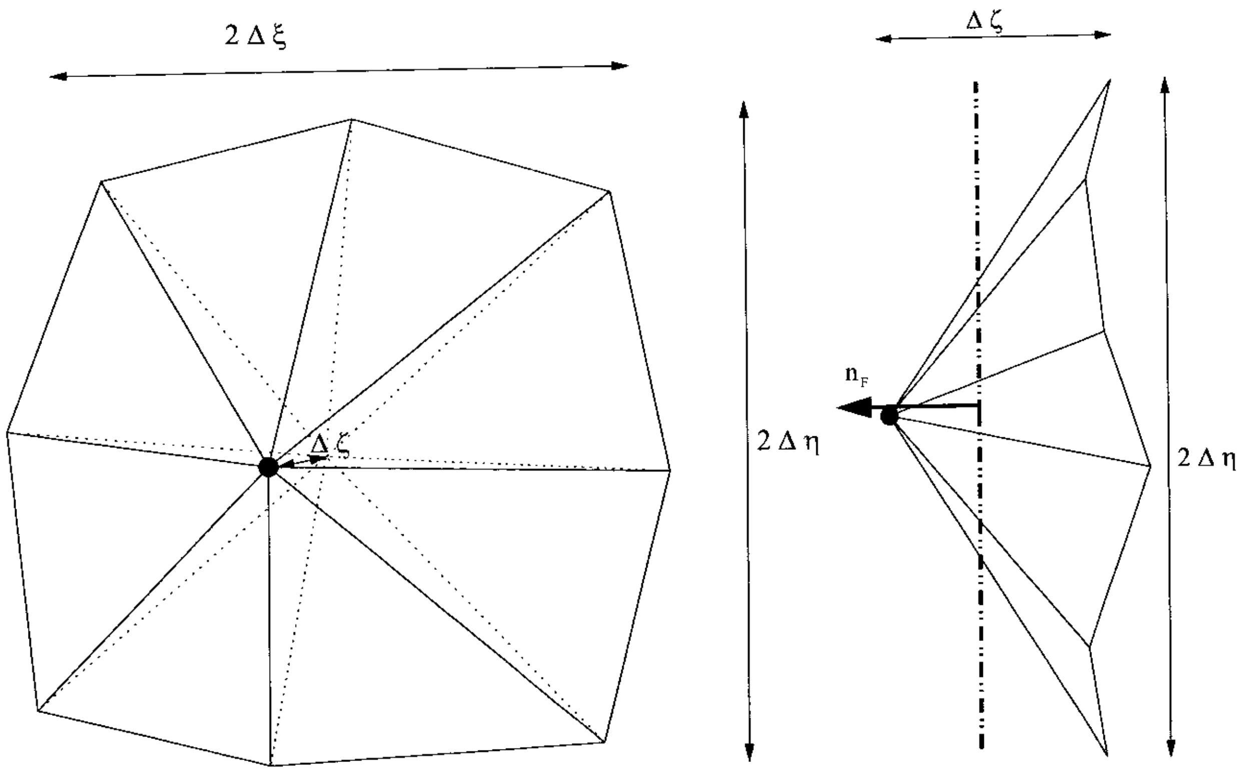


Figure 2: Face definition in local frame of reference

3.2 Error distribution for current scheme

The numerical scheme used in this work is an explicit upwinding scheme using artificial compressibility on arbitrary shaped finite volumes. The control volumes are m-faced, node centred polygons, and the faces are n-edged surfaces, defined by triangular segments. Figures 17 and 18 shows a representation of the faces and control volumes. More details about the scheme can be found in Wright and Turnock [29].

By separating the computational algorithm into its individual procedures a discretisation error for the numerical model and a distortion error for the geometrical entities involved can be determined. Table 1 lists the procedures in the algorithm, and the numerical discretisation and geometrical distortion errors. Figure 2 details the local frame of reference used for this analysis.

Algorithm operation	Mathematical operation	Discretisation error	Distortion error
Domain discretisation	$V_{CV} = \sum V_{tetra}$	–	0
	$A_F = \sum \lvert \vec{n}_F \cdot A_{tri} \rvert$	–	$\epsilon_A \propto \triangle\xi\triangle\zeta + \triangle\eta\triangle\zeta$
	$\vec{n}_F = \sum \frac{A_{tri}}{\sum A_{tri}} \vec{n}_{tri}$	–	$\epsilon_{\vec{n}_F} \propto \frac{\triangle\zeta}{\sqrt{\triangle\xi^2 + \triangle\eta^2}}$
Face flux	$\vec{F}_F = \frac{1}{2}\left(\vec{f}_R + \vec{f}_L\right) - \frac{1}{2}\left(\lvert\vec{f}_R\rvert - \lvert\vec{f}_L\rvert\right)$	$\epsilon \propto O(\triangle x)$	$\epsilon_{\vec{f}} \propto \epsilon_{\vec{n}_F},\ \epsilon_{\lvert\vec{f}\rvert} \propto \epsilon^2_{\vec{n}_F},\ \rightarrow \epsilon_{\vec{F}_F} \propto \epsilon^2_{\vec{n}_F}$
CV residual	$R_{CV} = \sum_{j=1}^{m} \vec{F}_j A_j$		$\epsilon_{R_{CV}} = \sum_{j=1}^{m} \epsilon_{\vec{F}_{F_j}} \epsilon_{A_j} \propto \epsilon^2_{\vec{n}_F} \epsilon_A$
Local time step	$\triangle t = \mathcal{B}\, CFL \frac{1}{S_x\lambda_x + S_y\lambda_y + S_y\lambda_y}$	$\epsilon \propto O(\triangle t^4)$	$\epsilon_{\triangle t} \propto \epsilon_A \epsilon_{\vec{n}_F}$
Sum residual to state	$U^{a+1}_{CV} = U^a_{CV} + \triangle t \sum_{k=1}^{4} \alpha_k R_{CV}$		$\epsilon_{\triangle U^a_{CV}} = \epsilon\triangle t \sum_{k=1}^{4} \alpha_k \epsilon_{R_{CV}} \propto \epsilon^3_{\vec{n}_F} \epsilon^2_A$

Table 1: Break down of discretisation and distortion errors

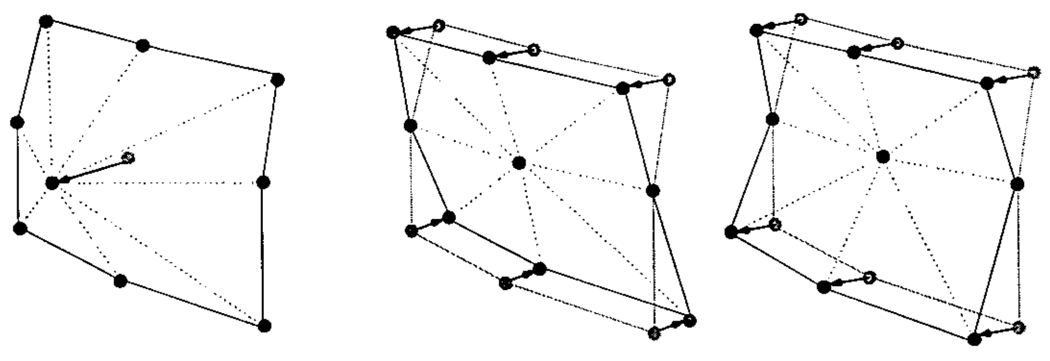


Figure 3: Types of facial distortion created to induce distortion errors in solution

3.3 Numerical study of flow solver

3.3.1 Error inducement

Discretisation errors and a number of distortion errors are present in any grid by definition, but to analyse them in depth a range of values have to be obtained for the same problem. A range of discretisation errors for both case studies (a 10% arc hump in a channel and a NACA0012 foil at 8° angle of attack) were created by the use of a range of mesh densities.

Distortion errors however have not been varied in the modelling and hence require inducement and study. Three forms of distortion error have been devised that shall be applied to the 'base' meshes used and for the discretisation error analysis. These three types of movement are shown in Figure 3 and can be defined as:

Planar movement of the central node on a face The central node on a face is moved in the direction of the face normal, thus increasing deviation from planarity, altering face area and altering cell volume. Other face properties will be altered to a minimal extent.

Misalignment of face normal with node to node vector All the nodes on the face are moved in a planar manner pivoting on the central node of the face. Thus the face normal will change, and deviated further from the node to node vector. All other face properties will have minimal variation, although faces using the same nodes will move the said node in different directions, hence causing a greater range of facial properties than initially desired.

Asymmetrical misalignment of face normal with node to node vector All the nodes except the centre node on the face are moved, in the same direction, in the plane of the face. This produces a 'V' shaped face. As a result the face deviates from planarity as well as having the face skew altered. facial area and cellular volume will also be affected.

3.3.2 Error analysis

The local error for each solution defined was calculated as the error in the pressure value for each node of the mesh when compared to the pressure obtained for the finest mesh . The average local error of the solution was therefore deemed to be the average of this local pressure error. A global measure of error was deemed to be the force upon a wall boundary of interest (the hump itself and the aerofoil for the two respective case studies). The forces obtained were compared to the validation data, and the error was deemed to be the difference.

The flow gradients of a given point were determined to be the average difference of the flow properties at the point and the surrounding control volumes. Thus average flow gradients could also be defined for a given mesh. Facial properties were calculated for all faces of a mesh, and the average, maximum, minimum and standard deviation for the mesh were determined.

In order to determine what relationship (if any) there is between a given parameter and the flow solution accuracy two methods of analysis were used. The average error for a given solution was compared to the relevant solution flow gradients and mesh geometrical properties on a global basis graphically. This highlighted overall trends. The second method was to perform a least squares fit between the local error and the local property, be it flow gradient or geometrical distortion). This second method allows a comparison between local and global solutions, as well as creating a template of the relative dependence of the solution upon the chosen parameters. The additional use of the least squares fit is that any given solution will have a combination of sources of error, hence a global analysis of the properties in isolation will not provide the true relationship.

In an attempt to separate the error due to geometrical parameters from the discretisation error, the 'base' solution error for each mesh density was removed from the solution error of the distorted mesh leaving the error due to distortion. Where average local errors for such meshes are listed it is the average of this difference in errors.

$$\epsilon_{MESH} = |\epsilon_{DISCRETISATION}| + |\epsilon_{DISTORTION}| \tag{1}$$

Figure 4 shows that the local error in the mesh varies linearly with mesh discretisation, as would be expected of a first order spatial discretisation code. The values 'tail off' towards lower H_1/H_i values because the error results are taken with respect to a finite solution $(H_1/H_i = 1)$.

Figure 5 shows the convergence characteristics of the flow code for increased mesh density, and indicates the necessity of a mesh dependence study to gain trustworthy results as well as the variation of results with a mesh of the same density but varying distortion. While the levels of distortion induced in these meshes was high, it is evident that mesh distortion has a large influence upon the solutions accuracy. Hence a purely mesh density based convergence study is insufficient to produce results of high quality and repeatability.

Finally, the least squares fit presented in Figure 6, strongly suggests that facial skew is by far the most important parameter to consider when reviewing solution error for both irrotational and rotational flows. Facial aspect ratio and misalignment of the face normal both vary with the mesh density, but the level of variance is only apparent for aspect ratio at higher mesh densities. Facial deviation from planarity has been defined as of much less importance for the hump flow than for the rotational flow around the foil.

The numerical tests carried out have led to the conclusions that :

- pressure gradients alone provide a suitable definition of flow gradients, and are more sensitive to solution error that velocity gradients;
- geometrical parameters do have a marked effect upon solution accuracy, more so at higher mesh densities;
- of the four geometrical parameters tested skew is the most important, with aspect ratios, misalignment of normals and deviation from planarity being influential in that order.

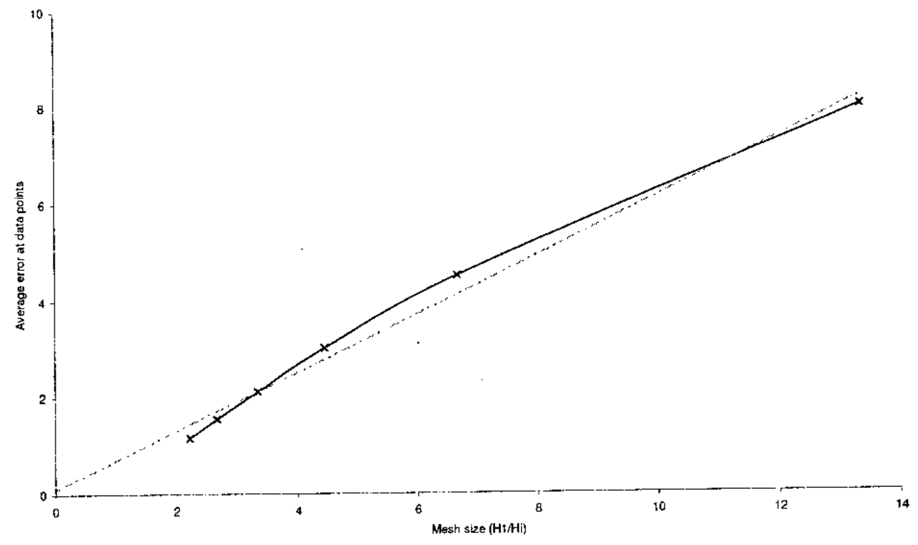


Figure 4: Variation in average local error with mesh discretisation for 10% arc hump

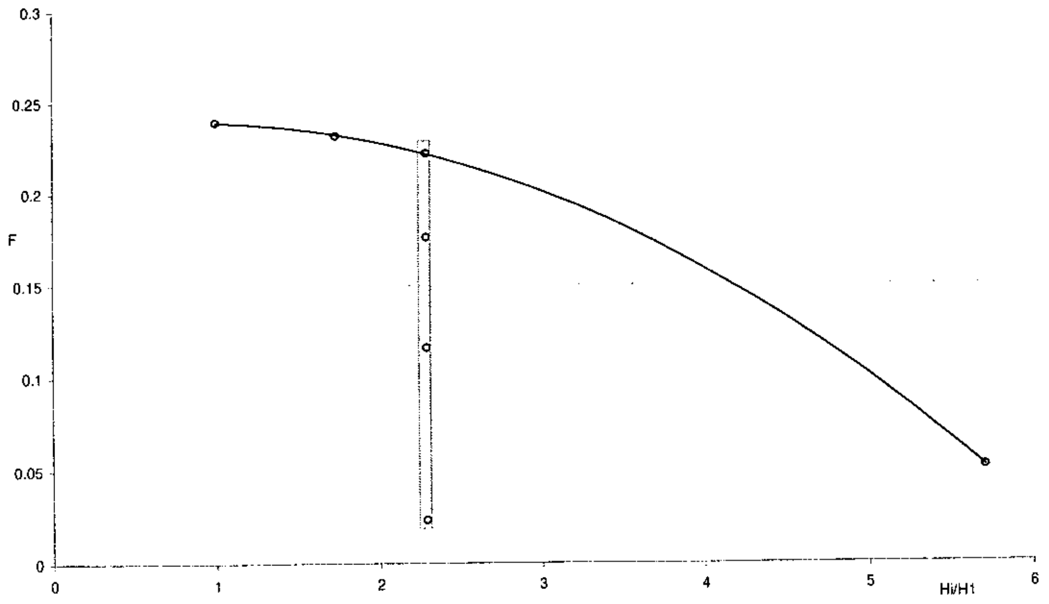


Figure 5: Error in force for NACA 0012 foil at 8° with mesh density variation

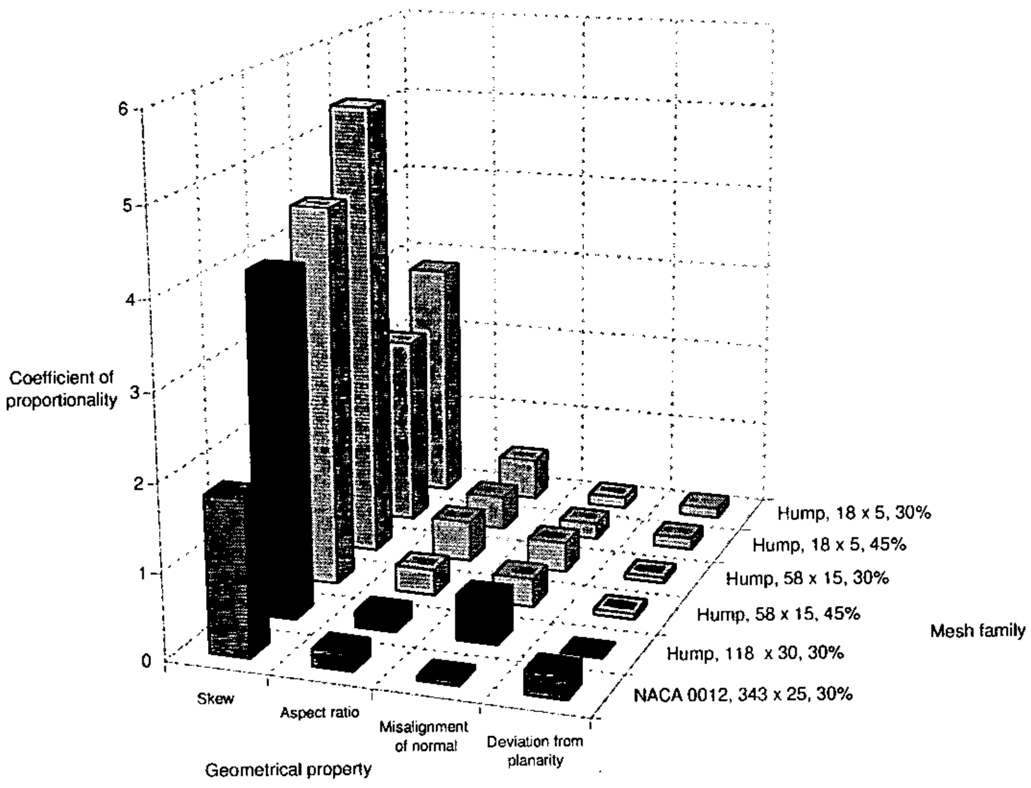


Figure 6: Relative importance of geometrical properties for all distorted mesh solutions

4 Grid and Data Structure

4.1 Introduction

As stated in Chapter 1 the physical domain must be discretised into finite volumes to allow a computational solution to be derived. It has been decided that arbitrarily shaped and sized control volumes will be the most efficient in the modelling of complex marine structures hence an unstructured meshing approach is to be used. In addition, the arbitrary nature of the entity topology and the allowance for adaptation requires a flexible and robust connectivity of the basic geometrical entities. The fast and efficient searching of the grid relies heavily upon the underlying data structure within which the geometrical entities are stored. Previous work (Rycroft [10]) created a complete connectivity data structure and investigated manners of improving efficiency of data storage, and it it this work that has been used as a starting point. Topics investigated are which geometrical entities are required to be defined, what level of connectivity is required between these entities and the manner in which they are stored.

4.2 Geometrical Entities

The discretisation of the physical domain into a grid is a geometric transformation. The methods used and the solutions obtained are entirely geometrical entities, thus any measurement of this grid must be in terms of geometric definitions. The base entities of a mesh, in order of hierarchy, are nodes, edges, faces and cells. That is, the most basic entity is a node. A series of nodes together can create edges; a series of edges defines a face; and a series of faces defines a cell. Figure 7 shows these four entities Solutions are limited to a finite volume of space whose extents are defined by specified bounding surfaces. It is within these surfaces that the grid needs to be created.

The majority of CFD solvers operate from a Eulerian point of view (fixed co-ordinate system with fluid moving through it), but a number operate within a Lagrangian framework (volume moving with the fluid) or a joint Lagrangian-Eulerian scheme[1]. The result of such schemes is a velocity associated with the mesh entities.

[1]termed ALE - Arbitrary Lagrangian Eulerian

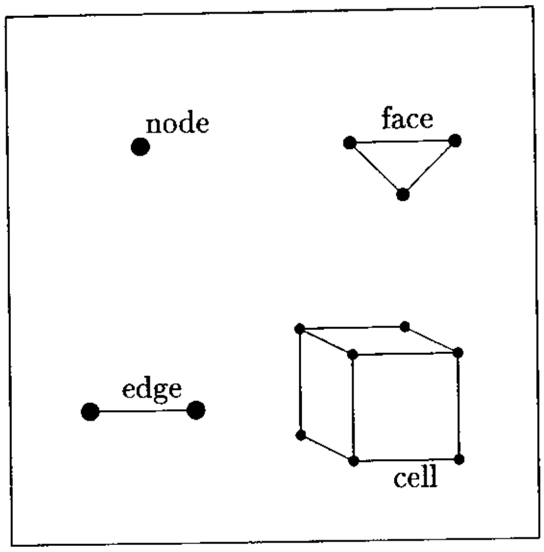


Figure 7: The basic geometrical entities

It is important to note that for a numerical solution to be obtained the mesh must adhere to the Global Conservation Law (GCL), which states that -

$$\frac{d}{dt}\int_V dV = \int_S W_S \, dS \tag{2}$$

V - cell volume W - boundary velocity S - cell face area

This can be broken into two separate aspects, volume conservation and surface area conservation. The volume conservation law (VCL) requires that the volumetric increment of a moving cell must be equal to the sum of the changes along the surfaces that enclose the volume, while the surface conservation law (SCL) states that the cell volume must be closed by its surfaces. Contravention of the surface conservation law will lead to a misrepresentation of the convective velocities and numerical violation of the volume conservation will produce extra sources or sinks in the physical media (Hindman [2], Demirdzic & Perić [32]). Such violations of the GCL place severe restrictions on the numerical solver.

4.3 Mesh Generation

When CFD began to be a recognised field of computational science a quarter of a century ago it was expected that grid generation would be a "solved" problem, with standardised solutions. However, the fact is that grid generation is a major pacing item in regard to the widespread use of CFD in design applications [33]. Cosner in 1995 noted that " a satisfactory grid can be developed to model nearly any configuration of interest. The issues at present focus on operational concerns such as ... quality" [34].

Traditional approaches to mesh generation, suitable for the application of finite difference techniques, have focused on the generation of hexahedral meshes meshes which conform to a transformation between Cartesian and Curvilinear co-ordinate systems.

The adoption of the finite volume and element methods have permitted tetrahedral, and more generally, polyhedral meshes to be utilised. The advantages of such methods are the greater flexibility that the mesh generation processes provide over traditional techniques, and the increased automation that may be installed. In addition, such unstructured meshes are intrinsically adaptive, a feature which has been subsequently exploited to dynamically manipulate the mesh throughout the solution process.

Recent advances in the construction of unstructured meshes, motivated by increased geometrical flexibility and viscous flow resolution, have lead towards hybrid meshes, constructed using more than one cell topology .

4.3.1 Structured, unstructured and hybrid meshes

Structured Meshes Meshes in which the connectivity between geometrical entities is implicit are termed structured meshes. This implicit connectivity is generated by the direct transformation from the physical Cartesian domain (x, y, z) and the computational domain(i, j, k). The two main groups of meshes belonging to this family are Cartesian and structured meshes.

Cartesian meshes are constructed directly in the physical space, with vertices aligned with the Cartesian axes. The geometrics are identical for all cells, and do not have to be discretely evaluated for each vertex, hence alleviating the need to store co-ordinate data whilst permitting efficient solver algorithms. This method can be expanded for complex geometries using a cut-cell approach, where the geometry is superimposed on a Cartesian mesh. Cells intersecting the solid boundary are cut and the region inside the geometry discarded. Clark et at [35] and Quirk [15] have demonstrated the ability of this technique to accurately model flow over complex shapes.

Structured meshes are produced by a transformation between the physical domain, described by Cartesian co-ordinates (x, y, z) and a computational domain, described by the generalised curvi-linear co-ordinates (ξ, η, ζ), where the boundaries of both domains coincide. The transformation allows the introduction of localised mesh clustering, such that the cells are non-regular in shape and orientation. This in turn requires that vertex co-ordinate information is stored. The connectivity of the mesh is however maintained uniformly. There are three basic topologies allowed in such meshes; the 'C', 'H' and 'O' meshes. The type of mesh that conforms most accurately with the physical domain tends to produce the best quality mesh. Figure 8 shows these meshes for the physical domain of an aerofoil.

The use of a singular transformation for complex, multi-object physical domains result in unsatisfactory physical meshes typically containing highly skewed cells.

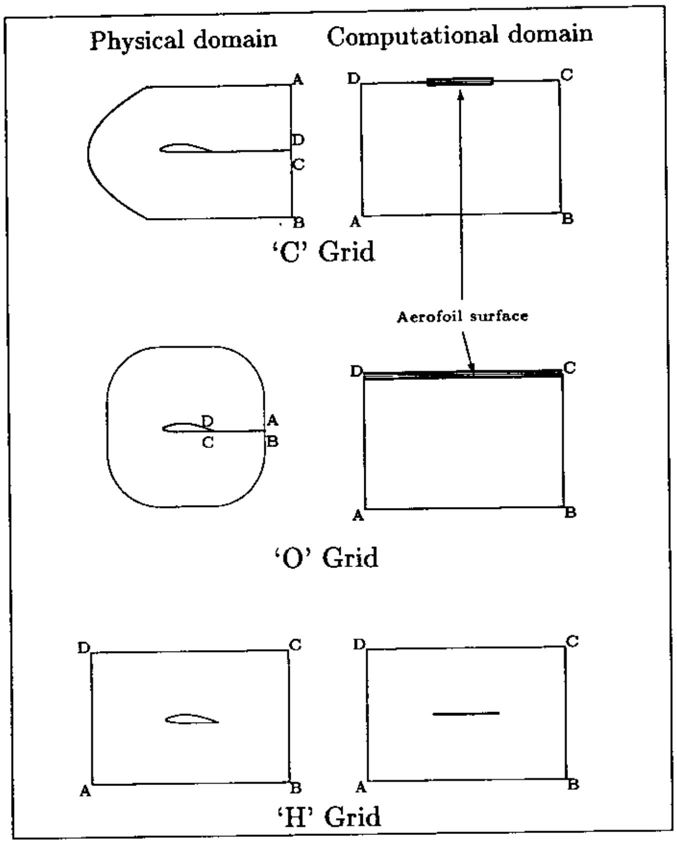


Figure 8: Physical and computational domains for structured grid topology

This considerably limits the complexity of the physical domain for which this method may be used. The multi-block approach extends the structured method by decomposing the domain into a series of sub-domains, termed 'blocks'. A structured mesh can then be generated within each of these blocks in isolation, conforming to local constraints only. The union of the blocks is not constrained by the physical to computational domain transformation, but does require additional connectivity constrains. Complex geometries can be modelled by decomposition and local meshing in this manner. An example of this is the work of Beddhu et al [36], which is reproduced in Figure 9. The high quality mesh on the hull surface can clearly be seen, but this density of mesh points upon the surface leads to an excess of data points in the far field.

The union of the blocks in the flow solver may be achieved in one two ways. The first is to ensure mesh consistence on the block faces, while the second does not require mesh compatibility, but instead relies upon interpolation routines to communicate information between blocks. Smoothness across the block boundaries is often a major hindrance upon overall grid quality. An increasingly popular solution to this is to allow the blocks to overlap, and hence use the numerical interpolation

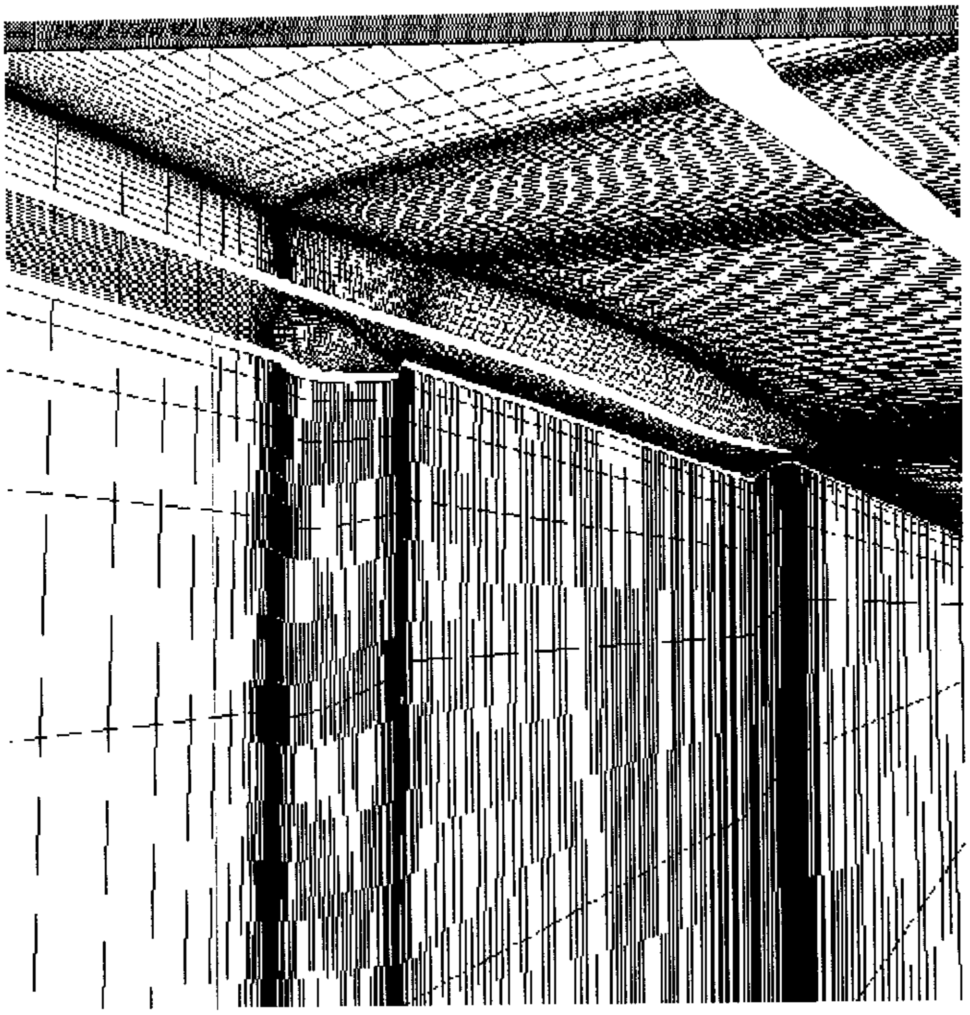


Figure 9: Grid structure for seven block DTMB Model 5415 grid. Reproduced with permission from [36] Copyright ©1998 The Author

method of information transfer. Lin et al [37] and Masuko [38] are two examples of the use of this method to model complex marine geometries. The drawback of such methods is the presence of redundant data points and the complex interpolation required between points if more than two meshes overlap at a give position [37]. A further alternative to these meshes are octree meshes where the initial mesh is subdivided, in a binary fashion in the vacinity of interesting flow features. Such methods do capture flow features more accurately while retaining a degree of the efficiency and low memory requirements of structured meshes. However the user is still limited by the manner in which extra data points are added and the inclusion of hanging nodes [18, Chp 21].

Unstructured Meshes

In unstructured meshes there is no implicit knowledge of the mesh connectivity. The connectivity between geometrical entities is required to be explicit, resulting in an increase of required memory and lowering in efficiency of data management. These additional costs are balanced by a vast improvement in the flexibility of the mesh and its ability to adapt. As a comparison to Figure 9, an unstructured mesh for the same hull form is shown in Figure 10, again reprinted with permission [39]. The total number of elements in the mesh is approximately a third that that of the structured mesh (507 744 tetrahedra compared to 1.44 million grid points), but the surface mesh distribution is comparable.

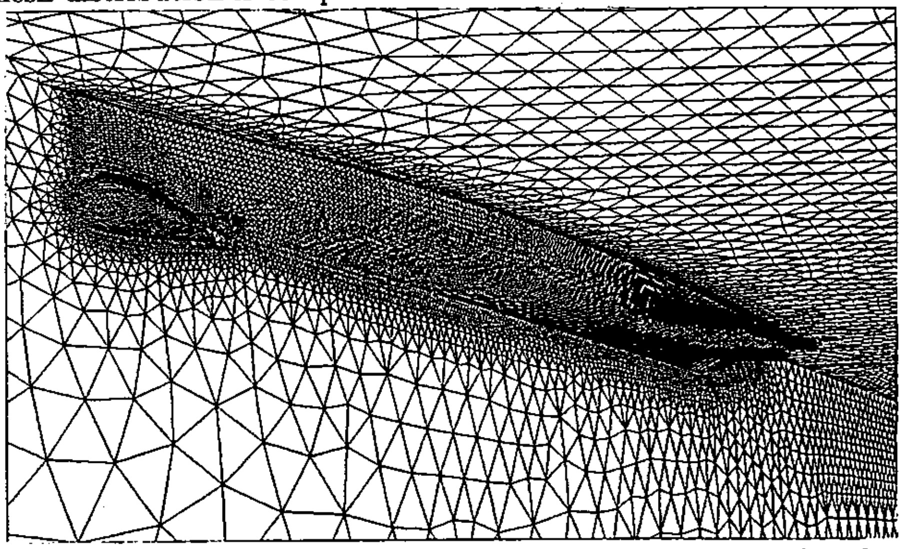


Figure 10: Unstructured grid for DTMB Model 5415 grid. Reproduced with permission from [39] Copyright ©1998 The Author

Unstructured meshes have become synonymous with triangular and tetrahedral control volumes since their introduction. However, as the only definition of an unstructured mesh is the lack of implicit connectivity, any topology of control volume can be used.

The forms of creating tetrahedral unstructured meshes are amply detailed elsewhere (see Rycroft [10]), and hence will not be discussed in this work.

The extra computational cost of the unstructured approach, as previously stated, is present in two forms. Firstly there is the extra storage required for the explicit connectivity between control volumes and geometrical entities, which is normally achieved using matrices describing the mapping of a particular cell topology to its vertices. Each matrix is thus a list of nodal identities or, vice-versa, a list of cells to which a vertex is connected. The additional memory cost can therefore be taken as the cost of these matrices. The 'real' cost however is much harder to clearly define, and is to be expected to be smaller. The use of tetrahedral meshes, and unstructured meshes in general, permits a much more efficient placement of points, and will normally attain the same solution resolution for a lot fewer control volumes or a higher resolution for the same solution degrees of freedom.

The second disadvantage of unstructured meshes is the increased data placement/retrieval time, due to non-direct memory addressing (e.g. operating on a vertex via the surrounding faces) and poor cache use. The inefficient cache use results from non-contiguous memory storage and non-optimisation of the algorithm with respect to the hardware. In comparison, structured data is arranged in a manner which can be exploited to maximise the efficiency of cache usage by grouping the entities that will be accessed simultaneously.

An additional problem, with is often overlooked, concerns the solver implementation. The correct implementation of higher order algorithms requires an expansion of the numerical stencil. The connectivity of a structured approach permits an easy and natural extension of the present stencil. Such a stencil may not naturally be switched to an unstructured mesh for which local interpolations are necessary, as a rule, to reconstruct the technique. The data structure must accommodate searching within the mesh such that adjacent geometrical entities may be identified at low computational cost.

The final difficulty incurred with tetrahedral based volumes occurs in viscous problems. Highly stretched cells are required to gain accurate resolution of the viscous terms but the generation of such volumes is not easily attained. Indeed, Aftosmis et al [40] state that there is no improvement in solution accuracy using stretched triangulations over hexahedral elements and thus offer a much reduced efficiency. The natural development from this is the use of hexahedral volumes in areas of viscous flow and tetrahedra in the outer domain; a hybrid mesh.

Hybrid Meshes

Hybrid meshes are a sub-set of unstructured meshes and are defined through the use of more than one cell topology. The use of mixed cell topology can improve efficiency as a result of utilising cells with a sparser connectivity matrices compared to tetrahedra and/or to optimise the cell topology with the flow domain, as described above. An example of type of mesh that can be created using such a method is shown in Figure 11.

Weatherill [41] uses a combination of quadrilaterals and triangles to generate meshes over complex 2D shapes. Unstructured 'micro-blocks' were either inserted in areas of poor grid quality or structured blocks were embedded in to the global mesh. Shaw and Peace [42] developed this approach for three dimensions, introducing pyramid volumes at the interface of hexahedra and tetrahedra. Sharov and Nakahashi [43] use a tetrahedral background mesh as a basis for marching a triangular front away from the structured mesh surfaces. The intersections of the two regions are evaluated to form a non-overlapping mesh. Nakahashi and Kano, in a more recent paper [44], have developed this method further by incorporating mesh adaptation, using local gas density as the adaption criterion.

There is a form of crossover with multi-block meshes in hybrid meshes, with separate areas utilising the most appropriate local gridding technique, and connecting with other blocks at set boundaries. As with standard multi-block meshes, over-lapping blocks can be used. An example of this is the work by Gomez [45] on the modelling of the NASA space shuttle. Figure 12 illustrates the mesh that Gomez created. As before though, costly interpolation routines are required in the over-lapping regions.

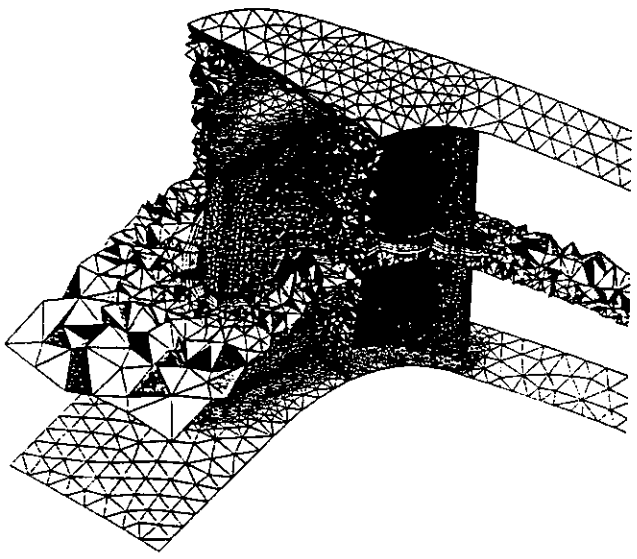


Figure 11: A surface and volume hybrid mesh for a turbine blade. Reproduced with permission from [18], Copyright ©1999 CRC Press

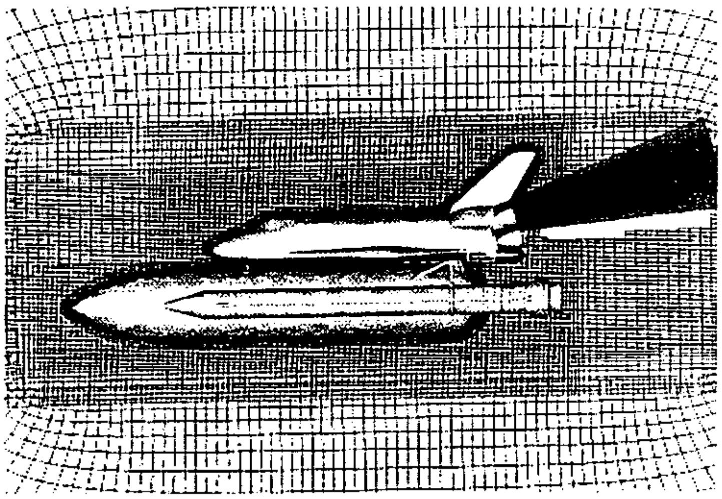


Figure 12: CHIMERA mesh around NASA Space Shuttle geometry. Reproduced with permission from [45], Copyright ©1994 AIAA

4.3.2 Cell centered and cell vertex schemes

The most commonly used method of mesh generation is to store the state vector information at the cell centre, as no extra geometrical construction and calculations are required. The main disadvantage of such methods is that boundary conditions require an extrapolation of the variables onto the boundary face. This causes an extra computational cost and, more importantly, in areas of high flow gradients the interpolation can introduce numerical inaccuracies.

When the flow variables are stored at the vertices of the mesh a larger flexibility of control volume shape is possible; cells surrounding the vertex may be amalgamated, creating overlapping control volumes, or a dual mesh can be created on top of the original mesh. The second option is currently a more common approach (Rycroft [10]). One of the advantages of the cell vertex approach is the accuracy with which boundary condition values can be defined, due to the positioning of vertices on the boundaries. A second advantage stems from the fact that loss of accuracy due to skewed cells is higher for cell centred schemes than cell vertex schemes [46][47]. The derived conclusion of this is that non-uniform meshes would benefit from a cell vertex scheme (Rycroft [10], page 58).

As accurate surface pressures are of prime importance in marine applications (free surface deformation and hull resistance for example) and taking into account the requirement of adaptable and robust meshes a cell vertex scheme has been chosen in this work. This scheme will operate upon unstructured arbitrary control volumes; that is each control volume will have an arbitrary number of faces and be identified in an unstructured manner. Further details of this method of control volume definition are detailed by Rycroft [10].

4.4 Data Storage Approach

The term *data structure* defines the relationships between the data items used by the computer program - in this case the geometrical entities. The algorithm of the program requires a variety of operations to be executed upon these entities and hence the manner in which their storage is organised will greatly influence the efficiency of the computer code as a whole. The decision to use a certain type of organisation of data structure depends upon many factors but the most relevant are the type of operations that are to be performed upon the data and the amount of computer memory which is available [18, Chp 14].

There are essentially two forms of data storage; sequentially stored *stacks* or *linked lists* where arbitrary sections of memory are joined by the storage of the memory position of the following record. These two forms of storage can be used to create linear lists, hash tables and tree structures. A graphical representation of these two structures can be seen in Figure 13.

Stacks are one dimensional arrays of contiguous memory, and can be used to associate data with a given index in a similar manner as the implicit location of

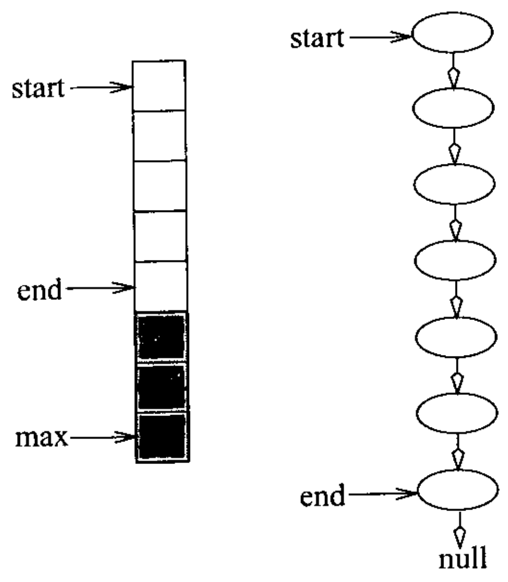


Figure 13: A graphical representation of a stack and linked list memory structure

data within a structured mesh. Stacks are however limited to one dimension and can hence only be used to describe the information in a one dimensional list, such as those of nodes. Links between separate stacks can be generated by storing the stack indices of relevant geometrical entities. For example a face entity may store the stack indices of the nodes that define its limits, thus allowing nodal data to be accessed via the face. The number of stacks and the type of inter-links is clearly prescribed by the type of data structure employed. Stacks store elements in a contiguous memory location which lends itself to efficient memory cache use. The main disadvantage of stacks is that they can not be manipulated easily, requiring a global reset if the number of records exceeds the stack size. Stacks can be created oversized, thus allowing growth and reduction in the number of records, but this does create unused space, hence the range of allowable ratios of actual array size to maximum array size tend to be critical. There is also the unavoidable global reset of the list of records at some time in the process.

Linked lists are individual locations in memory which are linked, linearly *or* non-linearly, by memory pointers. Each link in the list is treated individually, hence the order of storage can be manipulated and individual links can be removed and added. These manipulation operators can be achieved without changing the memory storage locations, the pointers are merely re-arranged. The clear advantage to this system is its adaptability and ease and speed of manipulation. A disadvantage is the inherent non-contiguous nature of the storage which often leads to a reduction in the efficiency of cache management. In addition, if distributed memory architectures are being considered, the complexity of partitioning the data structure is substantially increased [10]. A third disadvantage, notable for large problems, is that more memory space per record is required due to the memory address inclusion.

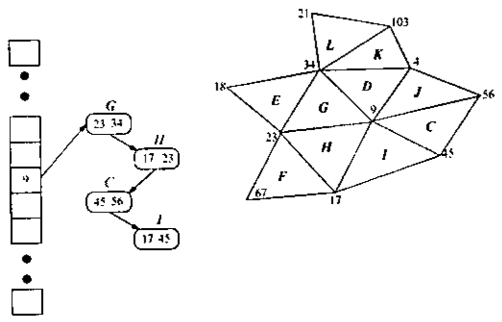


Figure 14: An example of a simple hash table, storing node ID connected to a triangular element

The final disadvantage, and with the most serious implications for overall algorithm efficiency, is that when global searches are initiated members of the list cannot be accessed directly; the operator is required to trace a path through the entire list, greatly reducing efficiency. This final consideration is often negated by the fact that direct addressing is not needed in many practical applications.[18, Chp 14].

These two forms of storage can be used to create linear lists, hash tables or tree structures. Linear lists are just that; one dimensional lists of data, most often in the form of a stack. Hash tables allow searching to be efficiently accomplished upon a list of data. For each given record the relevant entry in the table will contain all the entities that meet the appropriate criterion. For example, on a two dimensional mesh containing triangular elements, a hash table could be created that lists the nodes on a faces, with the indexing value being the node with the lowest (numerical) ID. The entry in the table will be the node ID, and the faces connected to that element where it is the lowest ID would be listed in some manner, as shown in Figure 14. There are many different forms and levels of complexity of such tables, and can be made highly specific to a given situation, using both sequential stacks and linked lists.

Tree structures are better suited for simple searches to ensure that an entity exists and range searches to find all data with set limits. A tree is a finite set whose elements are called nodes. The root node is the node from which all others can be accessed, and a leaf node is one without any further sub-divisions. Figure 15 demonstrates such a structure. Binary trees are ones where each node is linked at most to two branches, and are the most commonly used, but are by no means the only type. A node may have as many branches as it requires; quadtrees have four branches and octrees eight. These structures are often employed for two and three dimensional decomposition of a domain. A tree can be created via the use of linked lists or, in the case of an ordered tree, a stack. A binary tree will be equivalent to a doubly linked list, with pointers to the children. It should be noted however that pointers to parents can be employed equally as easily - the only disadvantage is the extra memory storage requirements.

Ideally for the final data structure used, a balance is sought which maintains the

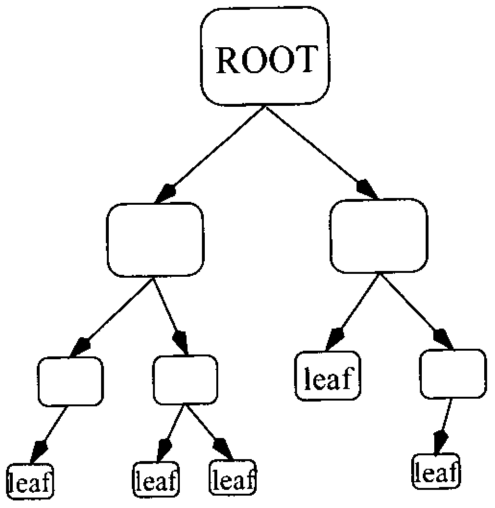


Figure 15: The representation of a simple tree structure

versatility of the dynamic allocation found in linked lists with the efficiency of data handling attained with stacks. Linear lists provide easily handled and retrieved data lists, while hash tables and tree structures allow searches to be efficient, as well as rapid sorting of the data. The case shown in Figure 14 shall be taken as an example. If **N** is the total number of nodes in the mesh, **F** the number of faces, and **R** the total number of records listed then -

- Memory requirements = $\mathbf{N}\mathrm{P} + \mathbf{F}(PTR + 2INT)$
- Operations for search = $\mathrm{O}(1 + \mathbf{R}/\mathbf{N})$
- Operations for adding record = $\mathrm{O}(1)$

where *PTR* and *INT* are the memory requirements for a pointer and integer respectively. The average value of $(\mathbf{R}/\mathbf{N})$ is generally small - ≈ 6 for a two dimensional mesh. If the data was stored in an array however, with **F** entries, each of which contain the three nodes that define the face, then -

- Memory requirements = $\mathbf{F}(3INT)$
- Operations for search = $\mathrm{O}(3\mathbf{F})$
- Operations for adding record = $\mathrm{O}(\mathbf{F})$

For a linked list of the same structure as the array then the memory requirements would increase to $\mathbf{F}(3INT + PTR)$ while the number of operations required for adding a record would drop to O(1).

The hash table demonstrated is clearly more efficient at retrieving the correct data but it is dedicated to only one type of search. Mode complex hash trees can have increased generality and thus allow a greater number of searches to be undertaken, but at the price of increased memory requirements.

4.5 Data Structure Implemented

The data structure implemented was developed from previous work within the Fluid Structural Interaction Research Group at the University of Southampton (see Rycroft [10], Turnock and Rycroft [48]). A cell vertex scheme has been utilised because of the increased accuracy in surface pressure modelling, as well as the fact that non-uniform meshes benefit from a cell vertex scheme [10, page 58]. The previous scheme has been proven to work for hybrid meshes of arbitrary topology but the memory overhead was high ($\approx .600\%$ with respect to a structured hexahedra mesh) and parts of the data structure were redundant for large sections of the computational process. The level of geometrical information, the nature of the mesh, the mesh connectivity and the nature of storage within the computer architecture have all been considered in a drive to improve the efficiency of the structure.

4.5.1 Arbitrary cell topology

Mesh adaption, as required if an optimal grid is to be developed during the solution phase, is one of the main drivers in the use of unstructured data types, and is often intrinsic in the data structure used. Unfortunately the use of hybrid meshes, with mixed cell topologies reduces the adaptive capabilities of the approach. Weatherill [41], as well as Shaw and Peace [42], explicitly define a limited number of control volume topologies for which their algorithms will operate successfully. This clearly limits the adaption process such that adjacent cells conform to a set connectivity matrix. A significant problem, particularly for cell centred schemes, is that of 'hanging nodes'. If a hexahedra and pyramid are adjacent to each other and the hexahedra is split into eight smaller hexahedra, then there is a 'hanging node' on the interface. This presents a disjoint in the mesh, where one control volume on an interface uses the node, but the other does not. Fluxes evaluated on the four adapted quadrilateral faces are not intrinsically equal to the flux through the base of the pyramid, and this may lead to a lack of conservation.

Ideally a data structure should be able to process control volumes, regardless of their topology. Rycroft [10] used the full range of geometrical entities, with complete sequential connectivity between them. Because of this complete connectivity, the ordering of individual elements by another entity was not important. As specified previously 4.2 there are four basic entities from which a mesh is created - nodes, edges, faces and cells - hence there were links from nodes to edges, from edges to nodes and faces, from faces to edges and cells and from cells to faces.

It should be reiterated that the cells are not the control volumes used in the solver, due to Rycroft's use of a node centred scheme, rather they are the cells of the initial mesh. The control volumes are created from the overlying node centred dual mesh, as shown in Figure 16. It can therefore be observed that the cells are generally redundant during the solution phase, used only for overall connectivity. The flow solver devised by Rycroft operated in an edge based algorithm, passing fluxes between nodes, the edges of the cell centred mesh defining the interface between

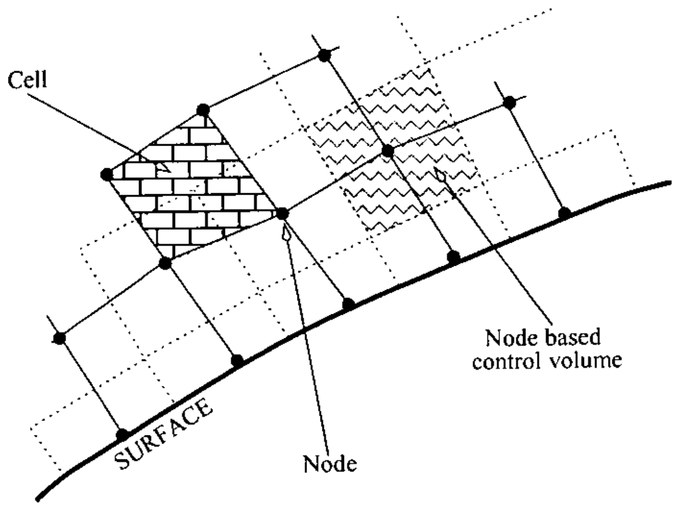


Figure 16: Cellular mesh, with over lying node-centered control volume mesh

node based control volumes.

Entity definition For the present work it was decided to operate purely with the overlying dual mesh. This decision was made because no justifiable benefit of the cell centred mesh could be established. As a results, the interfaces of the control volumes are actual faces of the mesh. Therefore it was decided that the flow solver would operate using an face based algorithm.

With the use of pointers to the nodes lying either side of the face, as well as node to node pointers edges have been made entirely redundant. The use of node to node pointers also allows complete mesh connectivity, and the face to node pointers allow spatial magnitudes to be defined. These two factors also result in cellular definition to be superfluous. Hence the scheme devised has negated the requirement for edge and cell definition. The only connectivity required is face to node and node to node.

Due to the unstructured nature of the data, and the arbitrary polyhedral shaping of each control volume the algorithm utilises triangular faces (henceforth termed *subfaces*) the union of which defines the faces of the control volume, as shown in Figure 17. The face area is a summation of the triangular segment areas and the facial normal is an area weighted average of the segment normals. This allows all geometrical properties to be calculated and the flux terms to be evaluated.

From these faces control volumes can be created, with the Global Conservation Law of surface area conservation being implicitly adhered to. The volume of the cells can be calculated from the triangular face segments, in an extension of the facial area calculation. Each triangular segment is defined as the base of a tetrahedron, with the apex being the control volume data point (i.e. the grid node). The summation of the volumes of these tetrahedra equals the volume of the cell. Figure 17 shows the planar area of a triangular segment and the volume subtended to it from one of the grid nodes adjacent to the face. It should be noted that this triangular segment subtends the grid node on the other side of the face, and hence a tetrahedral volume

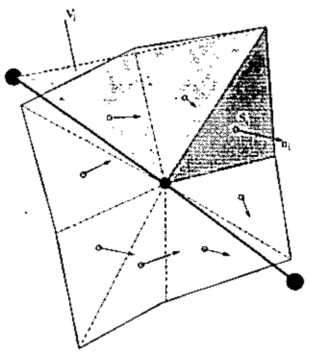


Figure 17: Control volume face definition

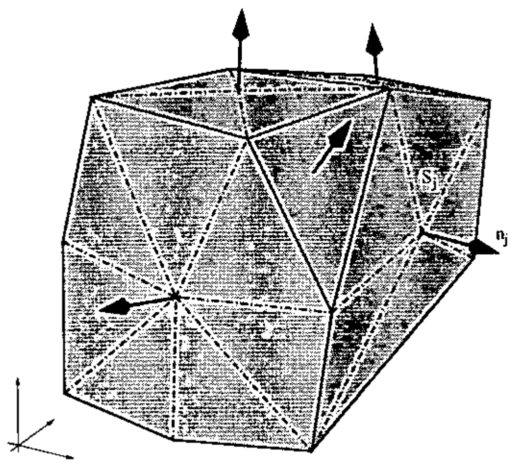


Figure 18: Polygon control volume

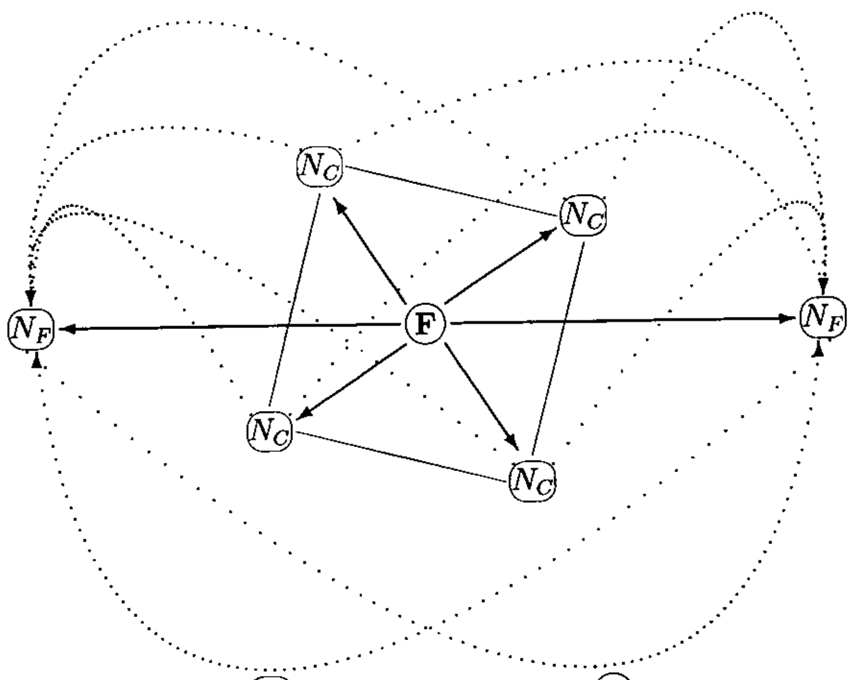
will have to be calculated for it as well.

If the mesh is constructed from a range of topologies there is no limit on the number of faces connected to a given vertex. Thus the control volumes associated with such a mesh may be complex polygons, as shown in Figure 18.

Mesh connectivity Because a cell vertex scheme has been chosen, the flux to/from a control volume requires that the face know which control volumes it is attached to.

To allow the construction of the face, and the calculations of areas and volumes, the faces have a connection to the 'construction' nodes that define the limits of the face. These construction nodes are in the position of cell and facial centres of the initial mesh. Figure 19 details all these connections graphically.

The construction nodes in turn require connection to the grid nodes (hence forth termed flow nodes) to allow correct interpolation of flow properties at boundaries but more importantly to aid the local remeshing process, and the subsequent reallocation of entity connections. In conjunction with this the flow nodes have connections to adjacent flow nodes to ensure the geometrical Global Conservation Law is maintained throughout mesh adaptation as well as aiding the efficiency of the remesh procedures.



N_F- Flow solver node N_C- Construction node **F**- Face

Figure 19: Connectivity of geometrical mesh

If the mesh were to be stationary and not subject to re-meshing during the solution then the node to node connections (including the construction node to node connections) can be removed. This would leave only face to node pointers defining the entire mesh connectivity.

Additional benefits of this structure are principally the ability to conduct a number of global searches efficiently without extra hash tables being created. Other benefits are the allowance for storage of extra information such as local geometrical properties and the reduced logic requirements for implementing boundary conditions due to the explicit inclusion of the face entity.

The in-house mesh generator *FLEXIMESH*, detailed in Ship Science Report 101, [49] and by Rycroft [10] was altered to produce files of correct format in both ASCII and binary format. Machine portable binary files were not generated due to the developmental nature of the code and the relatively limited number of architectures that the code was tested upon.

Memory requirements As an example to compare the varying levels of memory required by different approaches a structured mesh of hexahedra with N^3 points shall be used. As before , *PTR* signifies the memory requirement of a pointer, *INT* the memory space of an integer and *FLT* the memory for a float. The memory cost of all three types is normally identical.

Structured mesh The only data requiring storage are the spatial co-ordinates of the data points -

$$\text{MEMORY} = 3N^3 FLT$$

Unstructured hexahedra mesh In addition to the data point co-ordinates the cell to node connectivity is also required -

$$\begin{aligned}\text{MEMORY} &= 3N^3 FLT + 8(N-1)^3 PTR\\ \text{OVERHEAD} &= 8(N-1)^3 PTR\end{aligned}$$

Rycroft's arbitrary topology mesh The complete connectivity list devised would require each cell to list six faces, each one of those faces upon the cell to list four edges and those edges to list two nodes (hence $6 \times 4 \times 2$ connections)-

$$\begin{aligned}\text{MEMORY} &= 3N^3 FLT + 48(N-1)^3 PTR\\ \text{OVERHEAD} &= 48(N-1)^3 PTR\\ &= 600\% \text{ of standard unstructured mesh}\end{aligned}$$

New arbitrary topology mesh Because this mesh operates upon node centred control volumes the equivalent mesh was deemed to be one with $(N-1)^3$ control volumes; this specifies N^3 construction nodes and $(N-1)^3$ flow solver nodes. Each face will list six nodes (four construction nodes and two flow solver nodes), each construction node will list the surrounding flow solver nodes (eight construction nodes per control volume) and each flow solver node will list the surrounding flow solver nodes. In addition to this the spatial co-ordinates of the flow solver nodes will also be required -

$$\begin{aligned}\text{MEMORY} &= (3N^3 + 3(N-1)^3) FLT\\ &\quad + (6 \cdot 3N(N-1)^2 + 8(N-1)^3 + 2 \cdot 3(N-2)(N-1)^2) PTR\\ \text{OVERHEAD} &= 3(N-1)^3 FLT\\ &\quad + (18N(N-1)^2 + 8(N-1)^3 + 6(N-1)^2(N-2)) PTR\\ &\simeq 437.5\% \text{ of standard unstructured mesh}\end{aligned}$$

While there is still a substantial memory overhead compared to a purely structured mesh ,$\simeq$ 1167% assuming floats and pointers have the same memory cost, there has been a clear improvement for the previous arbitrary topology structure. It worth noting also that if the node to node connections are removed then the overhead drops to 262.5% of the unstructured hexahedral mesh. This would be possible if mesh adaptation were not required or if more complex search algorithms were used during adaptation. As stated in Section 4.3.1, the true memory cost is likely to be considerably lower.

4.5.2 Hybrid memory structure

In conjunction with the development of the basic data structure, and inter- connectivity, code development has concentrated on determining the most efficient manner of accessing the data. A combination of one dimensional lists of data (stacks) and non-contiguous linked lists have been utilised with memory pointers to produce a structure that allows efficient access to data as well as minimising memory overheads. The manner of accessing the data structure is shown in Figure 20. Memory pointers are denoted by arrows, and stacks by segmented boxes. It should be noted that the stack of memory allocated to storing facial data is over-sized, to allow for the creation and destruction of faces during grid adaptation without costly global reallocation.

The use of linked lists for the storage of all entities was deemed to be essential when considering the level and regularity of adaptation to be undertaken. The faces and nodes have also been given direct access from the array of pointers for ease of use. Direct access to the faces is required during the solver algorithm (which is facial based) and direct access of the nodes greatly benefits the remeshing process as described in Chapter 5. The construction nodes never require sufficient direct access to warrant a stack of pointers. The vast majority of construction node use is via the faces, which point directly to the node in question.

4.5.3 Entity fission and fusion

A generic tree data structure has been devised for the flow node section of the data structure to allow flow solver node creation and destruction by remeshing. As a node is split, it is 'deactivated' (denoted by the graphical representation of a node being covered), and pointers to its 'children' created. These children are not direct members of the stack accessed by the overall grid structure, but via the parent node. Figure 21 demonstrates how this process can be repeated for several 'generations' of children. Faces do not need this structure, they are merely maintained in the linear linked list, with new faces added to the end, or removed as necessary.

4.6 Summary

Structured mesh generation requires complex manipulation in order to mesh realistic geometrical configurations, such as geometrical singularities and diverse geometrical features. Triangular based unstructured meshes enable efficient mesh construction but lack do not provide sufficiently accurate solutions for viscous flow regimes. Hybrid meshes overcome such problems by the use of different control volume topologies in different domain zones. These meshes however reduce the inherent adaptability of unstructured meshes via the restriction of cell types. The approach chosen for this work is the use of arbitrary polyhedral volume definition which incorporates the benefits of a hybrid mesh with the adaptability of an unstructured mesh.

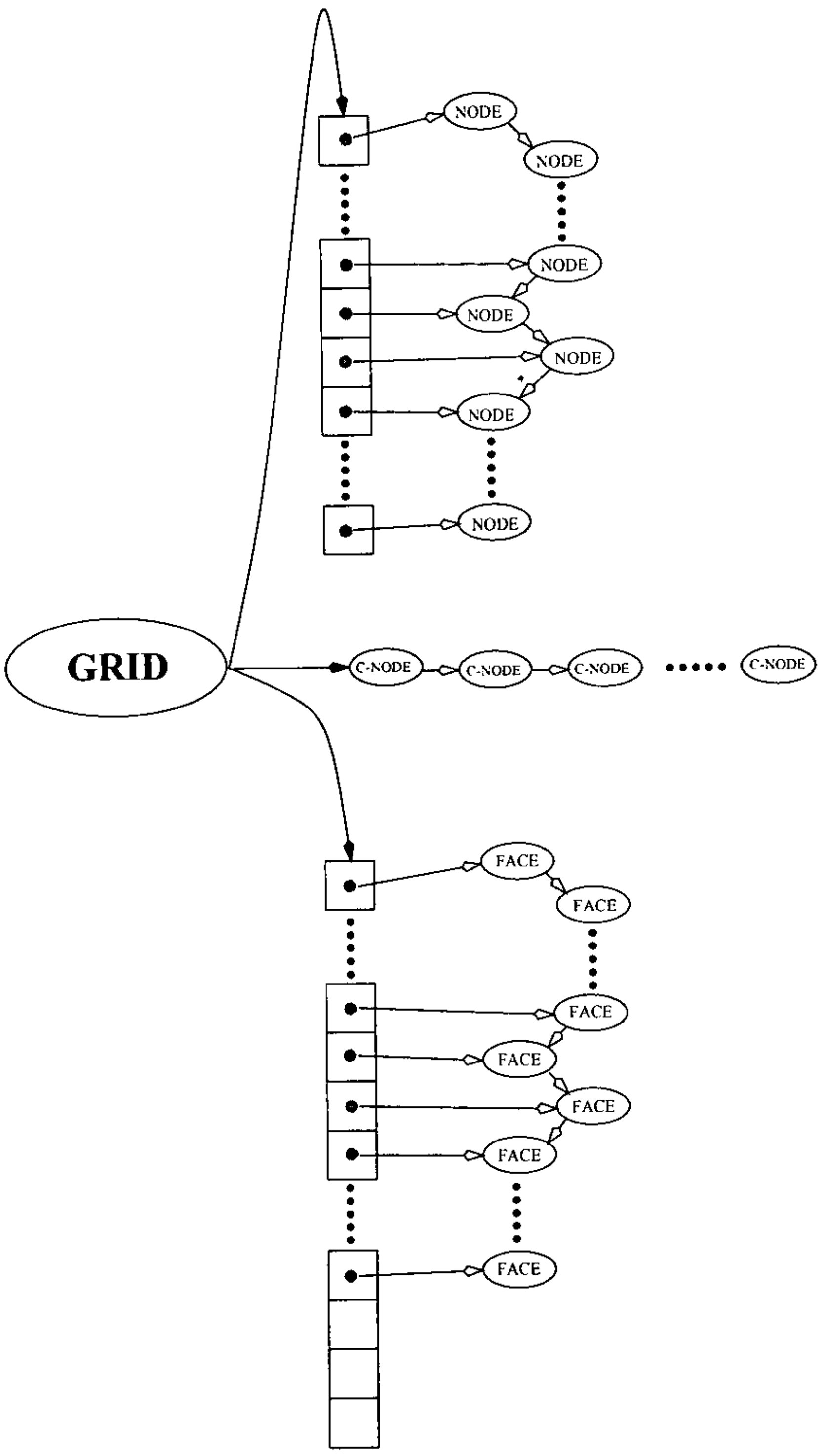


Figure 20: Data structure of geometrical mesh

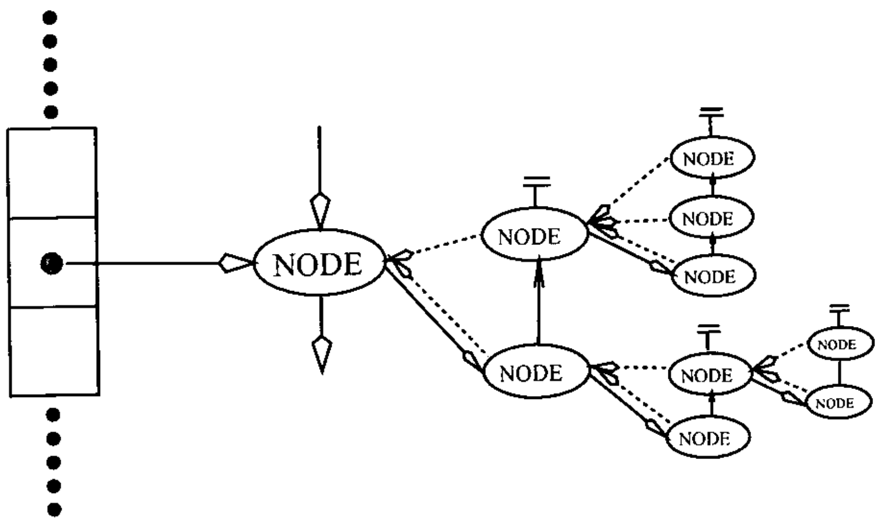


Figure 21: Tree data structure of flow solution nodes

A node based control volume approach has been adopted because of the importance of surface pressure definition in marine fluid flows. In order to accommodate the arbitrary polyhedra a data structure has been created that defines the mesh via geometrical entity definition and connectivity. This structure is 37% more efficient with memory use than previous such structures.

Increased ease of adaptation has been accommodated by storing entities in linked lists and using a tree structure to allow rapid cellular fission and fusion. Direct accessing of the node and face entities is achieved via the use of stacks of pointer to the linked lists.

Finally the underlying surfaces have been fully defined with bi-cubic spline patches, allowing accurate surface definition during successive mesh adaptations.

5 Control volume splitting

The reason for splitting a control volume is to lower the truncation error created due to the discretisation. As truncation error grows, so also do the flow property differentials between adjacent control volumes. Thus the control of this method has been chosen to be the local flow properties, namely pressure.

5.1 Geometrical Entity Based Approach

The first approach to splitting an arbitrary polyhedral was based upon selecting a face that was to be split, and then finding a face connected to the first face in the correct direction. This process was continued until a path had been traced around the control volume faces. The selected faces were then split via defining the number of sides, and entities reconnected to define two control volumes. Algorithm 1 details briefly each main step in the process. The process was set inside a loop containing all the volumes requiring splitting.

Algorithm 1

```
• Define cutting plane normal
• Define edge that is tangential to cutting normal
    1. Find the faces that are connected to that edge
    2. If more than 2 faces in list define arrangement
    3. While new edge is not original edge
        (a) Define edge opposite current edge
        (b) Find other face(s) attached to new edge
        (c) If more than 2 faces in list define arrangement
        (d) Make the new edge the current edge
• For each face in cutting list
    1. Define corners of face
    2. Define mid-point of side upon which cutting edge exists
    3. Creation of child faces
        (a) Define number of subfaces on original face ⇒ Both child
            faces will have same number of subfaces
        (b) Assess order of edges on faces ⇒ Insert new nodes in
            correct place
    4. Ensure new nodes and edges are not repeated on other faces
       to be cut
```

- Resetting construction nodes
- Resetting node to node connections

This method met with some success, and proved to work for most cases for a number of levels of adaption. A example of this was the adaptation of the mesh around a Wigley hull form, as detailed in Wright [50].

Figure 22 show the volumes of unacceptable mesh quality, as defined by geometrical calculations *before* any flow solution calculations have occurred. As can be seen, in this multi-block style generated mesh that geometrical variance is closely linked to the block-to-block geometry. Figure 23 shows the adapted control volumes and subsequent control volumes created (most have been split); the number of control volumes falling short of the required standard fell from 368 (3.2% of the initial total of 11448 control volumes) initially to just 96 within one level of adaptation. It should be noted that because of the present simplified manner of cell fusion (merely re-joining children of the same parent) the grid can not be made any more coarse than the initial grid. However, with the convection of flow solver nodes, the initial volumes in the extremities of the domain will, over time, develop an equivalent grid with very sparse meshing in the corners of the domain and dense node clustering in areas of interest. These areas tend to be the zones of high geometrical curvature and high flow property gradients. The volumes of grid quality outside the set limits after one level of refinement can be seen in Figure 24; clearly the volumes of low geometrical quality have been reduced.

Table 2 confirms this visual interpretation with numerical values. The standard deviation has significantly dropped, as has the maximum (relating to low quality).

Mesh	Minimum	Mean	Maximum	Std Deviation
Initial	0.034614	1.21657	6.946052	1.003025
Adapted	0.034614	1.21376	4.926340	0.917516

Table 2: Grid quality of Wigley hullform mesh before and after adaptation

However, problems start to appear with this algorithm due to the arbitrary nature of the control volumes and the allowance by the algorithm for the production of polyhedral volumes. Particular areas of concern were those of defining edge sides and corners; a generic definition of a 'corner' that would operate for a polygonal face proved troublesome for example. As entities become more and more abstracted from regular, symmetric shapes the algorithm has greater and greater difficulty splitting the faces in a symmetric manner, and the path which is 'traced' around the control volume does not necessarily meet with the starting point. Indeed, after ten levels of adaptation the algorithm became insufficient and broke down, leaving un-connected construction nodes and faces. Attempts were made to cover such exceptions but the code became extremely protracted, and it was felt that as arbitrary polyhedrals

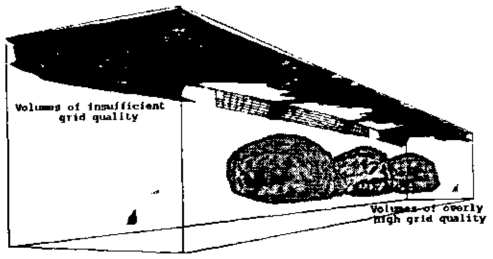

Volumes of insufficient grid quality
Volumes of overly high grid quality

Figure 22: Volumes with unacceptable quality in a mesh around a Wigley hullform

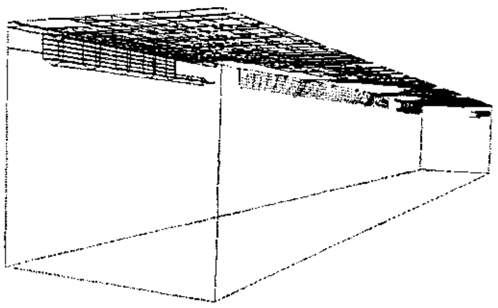


Figure 23: Adapted areas of Wigley hullform mesh

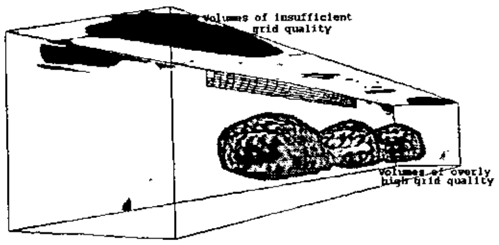

Volumes of insufficient grid quality
Volumes of overly high grid quality

Figure 24: Volumes with extreme quality in the Wigley hullform mesh after adaptation

were to be allowed there would always, eventually, be some case that exceeded the algorithm. Hence it was been decided that a more arbitrary splitting technique was required.

5.2 Geometrical Position Based Approach

The preferred algorithm was based upon the concept of a splitting plane. This passes through the volumetric centroid of the control volume, with its normal being defined by the geometrical grid quality algorithm. All the edges on each face attached to the control volume are checked to see if they pass through the plane. This produces a list of faces to be split, and the style in which they require splitting. Because of the non-reliance upon geometrical entity orientation this algorithm is reliable for successive generations of control volumes and degeneration of the volume topology. Algorithm 2 details each main step in the process for splitting one control volume. As with the entity based algorithm detailed in Section 5.1, the process will be set inside a loop, containing all the volumes requiring splitting.

Algorithm 2

```
• DEFINE CUTTING PLANE NORMAL

• DEFINE VOLUMETRIC CENTROID OF VOLUME AS POINT ON CUTTING PLANE

• IF (NODE ON BOUNDARY) ⇒ REPLACE FLOW NODE WITH A CONSTRUCTION
  NODE

• COLLECT FACES AND EDGES TO BE CUT

    1. FOR ALL FACES ATTACHED TO CONTROL VOLUME
       (A) FOR ALL INTERNAL EDGES ON FACE
             – IF (EDGE CROSSES CUTTING PLANE) ⇒ PLACE FACE AND
               EDGE IN CUTTING LISTS
       (B) FOR ALL EXTERNAL EDGES ON FACE
             – IF (EDGE CROSSES CUTTING PLANE) ⇒ PLACE FACE AND
               EDGE IN CUTTING LISTS
       (C) IF (NO EDGES CROSS CUTTING PLANE) ⇒ DETERMINE WHICH SIDE
           OF PLANE FACE IS AND PLACE IN LIST A OR LIST B

• ORDER EDGES AND LINK FACES

    1. COUNT FACES AND EDGES
    2. MARK START AND FINISH MARK FOR FACE IN EDGE LIST
    3. LIST REPEAT EDGES (ALL EXTERNAL EDGES MUST BE LISTED BY TWO
       FACES)
```

4. Check (and mark) parallel cut faces
   (a) If all splitting nodes are pre-existing
   (b) If all splitting nodes are external
   (c) If nodes are joined together in one continuous chain
   $\Rightarrow$ Mark face as to be a parallel cut

- If (number of faces > stack size) $\Rightarrow$ Reset face pointer stack
- Split the edges
  - Normalised position of cutting plane, X, crosses edge at -
    1. $0.0 \leq X < 0.25 \Rightarrow$ Use first node
    2. $0.25 \leq X < 0.75 \Rightarrow$ Create new node
    3. $0.75 \leq X < 1.0 \Rightarrow$ Use second node
- Reset the original faces and spawn new face. For each face
  1. Determine which side of cutting plane all construction nodes lie; place in appropriate list (list A and list B)
  2. If (face on boundary & open) $\Rightarrow$ Add flow node replacement construction node to appropriate list
  3. If (face is internal & enclosed) $\Rightarrow$ Remove central construction node from appropriate list
  4. Add splitting nodes for face to both lists
  5. Place nodes in order
     (a) Define angle of node with respect to first node in list
     (b) Sort list of nodes for angle $= 0 \rightarrow 2\pi$
  6. If (number of construction nodes in list ≥ 4) add central node
  7. Transfer all information into face structure. Original face reconstructed from list A, new face created from list B
- Reset tail of construction node list
  1. Mark first construction node in list
  2. Reset beginning of list to second member
  3. Place first member at end of list
  4. Set this to be the end of the list, not a looped list
  5. Store the ID of the construction node for use by the new data node

```
    6. Reset global ID of construction node

• Spawn the new flow node

    1. Set up new memory for child flow nodes
    2. Define characteristic length of control volume
    3. Count number of external faces
    4. If (number of external faces = 0) $\Rightarrow$
       New node position = centroid of new volume
    5. If (number of external faces = 1) $\Rightarrow$
       New node position = centroid of new external face
    6. If (number of external faces $\geq 2$) $\Rightarrow$
       (a) Find common construction nodes
       (b) If (number of common nodes = 1) $\Rightarrow$
           Convert construction node to flow node
       (c) If (number of common nodes = 2) $\Rightarrow$
           Make flow node $\frac{1}{2}$ way between nodes
       (d) If (number of common nodes = 3) $\Rightarrow$
           Convert construction node closest to average position
       (e) Remove converted construction node from list of con-
           struction nodes defining face if required

• Reorder nodes on faces

    1. Place nodes in order with respect to flow node

• Make splitting face

    1. Collect all splitting nodes
    2. Place nodes in order
    3. If (number of construction nodes $\geq 4$) $\Rightarrow$ make central
       construction node

• Reset status of flow nodes

    1. Flag parent as DEAD
    2. Flag children as ALIVE

• Reset the geometrical entity linking

    1. Reset face to flow node links
    2. Count number of construction nodes attached to flow nodes
    3. Reset construction node to flow node links
    4. Reset flow node to flow node links
```

5.3 Benefits of position based approach

This new algorithm has the advantage of only working upon the edge and node entities, not on the face as a whole. This reduces the number of possible situations greatly; an edge has direction, yet contains only two nodes, while a face can have any number of nodes great than three to define it, hence creating an infinite number of combinations.

The other major improvement of this algorithm is the use of placing the nodes in order by angle. Previous algorithms tried to maintain the existing node connectivity and ordering but this caused a great number of problems and produced verbose coding. Instead the use of a list containing the required construction nodes, and then sorting the list by angle has proven to be efficient in both length and complexity of code as well as CPU usage. The only variable in this method is the selection of a point that is defined as being the origin. It is for this reason that the reordering occurs twice. To define the place of the new flow node the construction nodes have to be in order, but the final order of the nodes can only be specified once the flow node position is declared. Thus the first sort places the construction nodes in cyclical order, and the second sort organises them with respect to the flow node.

To date the algorithm has proven robust and efficient, solving both two dimensional and three dimensional problems. Indeed, more problems have been encountered with auxiliary routines, such as grid quality definitions, after adaption than with the control volume splitting routines themselves. Validation and verification of the algorithm are on going.

5.4 Surface definition

Because the underlying concept of control volume splitting is increased accuracy for a given solution time, there is also the requirement for the geometrical definition to have similar accuracy. As Figure 25 shows, without knowledge of the underlying boundary surface definition boundary degradation will occur.

To maintain an accurate definition of the surface a separate definition was stored in bi-cubic spline patch form. The position of any entity upon the surface is defined by a unique ID and local parametric positions.

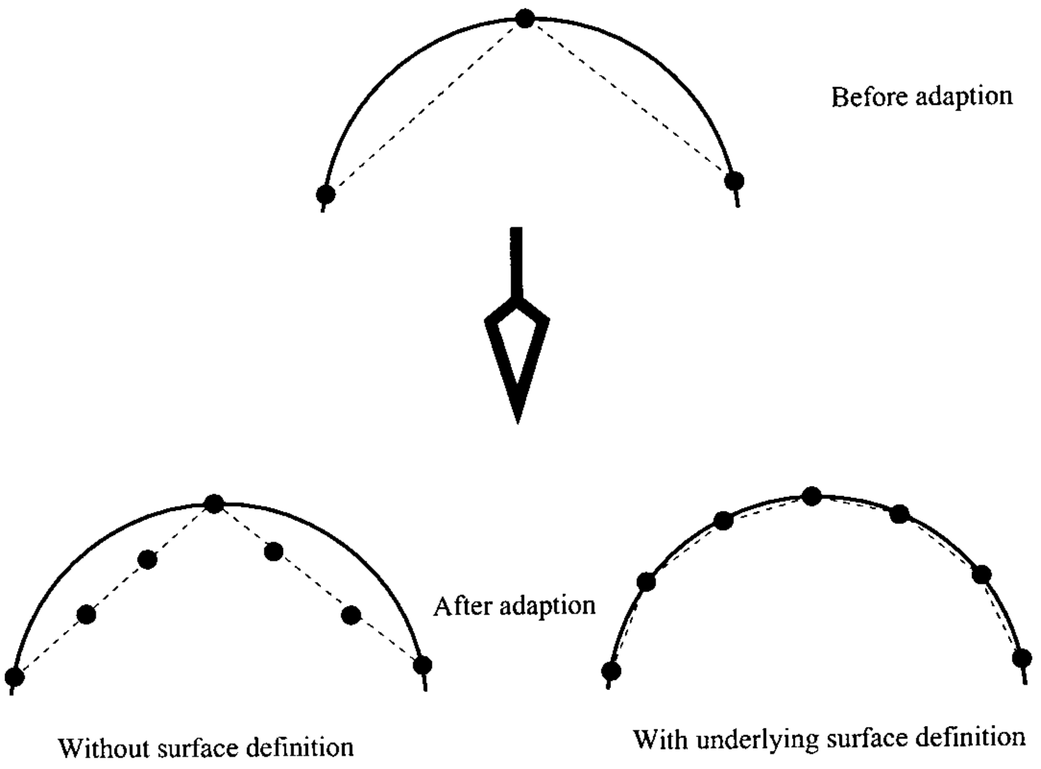

Before adaption
After adaption
Without surface definition
With underlying surface definition

Figure 25: Effect of surface definition upon mesh adaption

6 Control volume merging

The reasons for merging control volumes together are the exact opposite of splitting a volume; the local error is appreciably less than the norm of the local error over the domain. This means that the solution is being inefficient; the increased accuracy in this area will be negated by the surrounding lower accuracy and hence the CPU resources required for the area are being squandered. As with the control of the volume splitting, local flow properties have been utilised. When the pressure differential between two adjacent control volumes falls below a set limit (adjusted to the required accuracy level), then they are merged together.

6.1 Initial method of merging control volumes

To date the method of merging control volumes together is limited to those volumes that have previously been split from the same parent. While this is limiting, if the initial mesh is suitably coarse then it should not be restricting to the solution efficiency. The basic premise of the approach is to go back a level of the flow node tree data structure, flagging the children as dead, and the parent as alive. The code has been set up to allow eight levels of the tree to be in existence at any one time. Due to the use of a linked list within the tree structure a parent flow node may have more than two children. Hence one or more levels of children can be removed, and the link reset to the required level of children, if deemed necessary. Algorithm 3 details the approach used to merge volumes. As with the splitting algorithm, this will be placed inside a loop that contains all instances of merging required.

Algorithm 3

```
• FIND ALL CONTROL VOLUMES OF THE SAME PARENT AND GENERATION

• RESET STATUS OF FLOW NODES AND FACES
    1. FLAG CHILDREN AS DEAD
    2. FLAG PARENT AS ALIVE
    3. FLAG SPLITTING FACES AS DEAD

• RESET THE GEOMETRICAL ENTITY LINKING
    1. RESET FACE TO FLOW NODE LINKS, ALL POINTING TO PARENT NODE
    2. RESET CONSTRUCTION TO FLOW NODE LINKS, ALL POINTING TO PARENT NODE
    3. RESET FLOW NODE TO FLOW NODE LINKS, ALL POINTING TO PARENT NODE
```

As with the splitting algorithm, validation and verification is an on going task but results to date are encouraging.

6.2 Influences and effects

The most noticeable effect control volume merging has upon a solution is its convergence time. This can be reduced drastically, for the same accuracy of solution, by merging of control volumes in the far field.

It is would noting that efficiency improvement is not linear due to the maintenance of the present face structure; this results in more than one face between two adjacent control volumes. An algorithm to merge such groups of faces are being developed, but, as with the destruction of levels of the flow node tree, the instances of use is a subject of development. If faces are merged at the same time as the control volumes, then memory and solution time CPU are decreased but mesh adaptation CPU time is increased. The most efficient balance between merging faces and leaving the current structure is case dependant and not resolved as yet.

An important point to be noted about this methodology is that because of the allowance for more than one face between adjoining control volumes, and the inherent subdivision of the control volumes into tetrahedra, the problem of hanging nodes and subsequent loss of conservation are negated [10, page21].

7 Entity stretching and convection

While the redistribution of mesh nodes has been used to improve solution accuracy via re-location to areas of high flow gradients, it has been decided that the volume splitting and merging detailed in Chapters 5 and 6 will provide sufficient flexibility and range for accuracy requirements. Instead node redistribution has been used to improve solution convergence rates and stability. This has been achieved by redistributing the construction nodes according to the local grid quality assessment.

As listed in Chapter 3, skew, aspect ratio, deviation from planarity and misalignment of the normal are all assessed and combined, weighted with respect to Figure 6, to define the local geometrical grid quality. This quality rating has been given a limitor, and if a given volume has a value greater than this limit then two processes are initiated.

The first approach to improving the geometrical quality of the mesh is a movement of faces to improve orthogonality, hence lowering facial skew and misalignment of the normals. The second process involves movement of individual construction nodes to increase planarity. Algorithms 4 and 5 detail these two processes. Figure 26 shows a representation of a face, and details the labelling of entities.

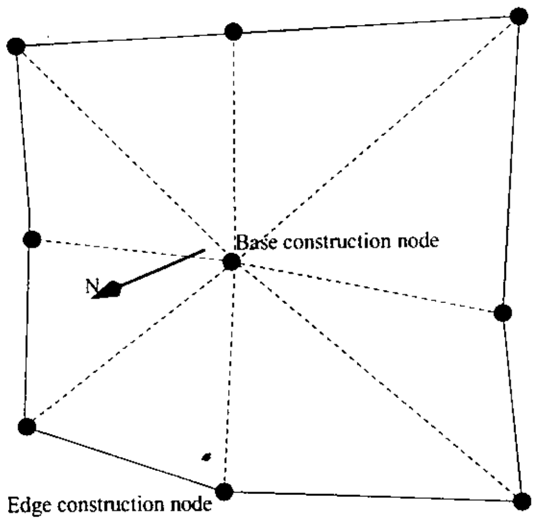


Figure 26: Node and face definition

Algorithm 4

- Define face normal (termed $\vec{n_F}$)
- Define flow node to flow node vector (termed $\vec{NN}$)
- Define vector A, passing through base construction node, defined as $A = \vec{n_F} \times \vec{NN}$
- Move construction nodes of face, rotated about vector A, until $\vec{n_F} = \vec{NN}$

Algorithm 5

- Define face normal, $\vec{n_F}$
- Using least squares fit, move base construction node to minimised deviation from planarity
- Redefine face normal, $\vec{n_F}$
- Move each of the edge construction nodes to bring it into plane $\Rightarrow P_{new} = P_{old} + \left(\vec{OP}_{old} \cdot \vec{n_F}\right) \vec{n_F}$

It can be observed that the relevant node movement to make one face planar may not be the correct movement to make a co-joining face also planar. However, provided the planes are not the same, they will cross and with the introduction of third plane a unique point that lies on all three planes can be found. In theory more than three plane can have a unique co-located point, but it is not a mathematical certainty. Thus when moving around the faces attached to a given control volume a variation of Algorithm 5 is used. Due to the developmental nature of the work and in aid of robustness, only the combination of two faces at any one given time are going to be considered. Algorithm 6 details the manner in which the node adjoining two faces is moved. For nodes attached to three or more faces, an iterative process is used, where the second face becomes the first and the next face becomes the second, until all the faces are considered.

Initial studies using these algorithms have quantifiably improved the grid quality. As yet the algorithms have not been used within a fully converged flow solution model, but studies are on-going and results to date are promising.

Algorithm 6

- DEFINE THE BASE NODE POSITION AS C
- DEFINE ORIGINAL NODE POSITION, P, AS P^0
- DEFINE 1st FACE NORMAL, $\vec{n_F^1}$
- $\Delta^1 = C - P^0$
- $\delta^1 = \Delta^1 \cdot \vec{n_F^1}$
- $P^1 = P^0 + \delta^1 \vec{n_F^1}$
- DEFINE 2nd FACE NORMAL, $\vec{n_F^2}$
- $\Delta^2 = C - P^1$
- $\delta^2 = \Delta^2 \cdot \vec{n_F^2}$
- $P^2 = P^1 + \delta^2 \vec{n_F^2}$
- NEW NODE POSITION, $P_{new} = P^2$

8 Conclusions and future developments

To improve numerical flow solutions without increasing the complexity of the mathematical model a method of mesh adaption has been devised. This has required developments in the data structure employed (in both definition and computer implementation) and the manner in which the mesh adaption has been implemented.

In order to fully utilise the available types of adaption an unstructured, arbitrary polyhedral control volume data structure has been created. This allows a more structured mesh in those areas that benefit from it and unstructured meshing in areas of complex geometry and in the far field where mesh degeneration allows increased efficiency of CPU usage. In order to accommodate this structure the geometrical entities are defined separately and linked via a connectivity list. This is a development of previous work [10], but the efficiency of memory storage has been improved by a factor of 1.4.

The geometrical entities have been stored in the computer memory using a range of data structured to allow easy access while maintaining a good level of flexibility. A tree data structure has been used for the flow node geometrical structure to allow control volume splitting and merging easily, as well as inherently allowing the expansion into multigrid solutions.

Mesh quality has been defined via the measurement of geometrical and flow properties, and analysed by least squares fitting. Pressure differential across volumes is used to control cell splitting and merging while a combination of the geometrical properties is used to control mesh orthogonality and planarity.

Algorithms are detailed for splitting and merging arbitrary polyhedral control volumes, and the requirement for a fully defined underlying surface definition is noted. Algorithms to remove deviation from planarity and improve orthogonality are also detailed, and the problem of how to locate a node joined to more than three faces to improve planarity is approached.

There are presently three areas that are being developed to increase the capabilities of the code.

Validation and verification : The primary goal at present is to conclude validation of the algorithms and verification of the computational code. There are still a number of cases, incorporating geometrical singularities and complex three dimensionally curved surface intersections, that have not been fully studied.

Incorporation of code within flow solver : The framework to incorporate the computational algorithms within the flow solver (detailed by Wright and Turnock [29]) is present, but as yet no adapted solutions have been gained. This is purely due to the on-going verification work.

Multigrid solutions : Once the adaption algorithms are incorporated successfully within the flow solver they shall be used to create multiple levels of finer meshes to allow increased efficiency via the use of multigrid techniques. This is seen as a relatively straight forward piece of work; the most complex section will be the correct transference of information between levels of the mesh while maintaining conservation of mass, momentum and energy.

References

[1] W. G. Habashi, J. Dompierre, Y. Bourgault, M. Fortin, and M. G. Vallet. "Certifiable Computational Fluid Dynamics Through Mesh Optimization". *AIAA Journal*, 36(5):703–711, May 1998.

[2] R. G. Hindman. "Generalized coordinate forms of governing fluid equations and associated geometrically induced errors". *AIAA Journal*, 20:1359, 1982.

[3] A. Ilinca, R. Camarero, J. Y. Trepanier, and M. Reggio. "Error estimator and adaptive moving grids for finite volumes schemes". *AIAA Journal*, 33(11):2058–2065, November 1995.

[4] J. Oliger and X. Zhu. "Stability and error estimation for component adaptive grid methods". *Applied Numerical Mathematics*, 20:407–426, 1996.

[5] A. Lawal. "adaptive grid method for convection-diffusion equations". *Int. J. Heat Mass Transfer*, 33(8):1633–1641, August 1990.

[6] Y. Tu and J. F. Thompson. "Three-dimensional solution-adaptive grid generation on composite configurations". *AIAA Journal*, 29(12):2025–2027, December 1991.

[7] J. H. Kwon and H. K. Jeong. "Solution-adaptive grid generation for compressible flow". *Computers and Fluids*, 25(6):551–560, June 1996.

[8] Y. Kallinderis. "A finite volume Navier-Stokes algorithm for adaptive grids". *International Journal of Numerical Methods in fluids*, 15:193–217, 1992.

[9] J. U. Brackbill and J. S. Saltzman. "Adaptive zoning for Singular Problems in Two Dimensions". *Journal of Computational Physics*, 46(2):342–368, 1982.

[10] N. C. Rycroft. *"An adaptive three dimensional, finite volume, Euler solver for distributed architecture using arbitrary polyhedral cells"*. PhD thesis, University of Southampton, 1998.

[11] N. K. Yamaleev. "Minimization of the Truncation Error by Grid Adaption". ICASE Report 99-46, NASA, November 1999.

[12] A. Bowyer. "Computing Dirichlet tessellations". *The Computer Journal*, 24(2):162–166, 1981.

[13] D. F. Watson. "Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes". *The Computer Journal*, 24(2):167–172, 1981.

[14] P. R. Eiseman. "Alternating direction adaptive grid generation". *AIAA Journal*, 23(4):551–560, April 1985.

[15] J. J. Quirk. "An alternative to unstructured grids for computing gas dynamic flows around arbitrarily complex two-dimensional bodies". *Computers and Fluids*, 23(1):125–142, 1994.

[16] C. Özturan. " Worst case complexity of parallel triangular mesh refinement by longest edge bisection". ICASE Report 96-56, NASA, 1996.

[17] A. Rassineux. "Generation and optimisation of tetrahedral meshes by advancing front technique". *International Journal for numerical Methods in Engineering*, 41:651–674, 1998.

[18] J. F. Thompson, B. K. Soni, and N. P. Weatherill, editors. *"Handbook of Grid Generation"*. CRC Press, 1999. ISBN 0-8493-2687-7.

[19] C. B. Allen. "Grid adaptation for unsteady flow computations". *Proc Institution of Mechanical Engineers*, 211, 1997.

[20] V. Parthasarathy and Y. Kallinderis. "New multigrid approach for three-dimensional unstructured, adaptive grids". *AIAA Journal*, 32(5):956–963, May 1994.

[21] V. Parthasarathy and Y. Kallinderis. "Directional viscous multigrid using adaptive prismatic meshes". *AIAA Journal*, 33(1):69–78, January 1995.

[22] D. J. Mavriplis. "Multigrid solution strategies for adaptive meshing problems". ICASE Report 95-14, NASA, 1995.

[23] K. Riemslagh and E. Dick. "A multigrid method with unstructured adaptive grids for steady Euler equations". *Journal of Computational and Applied Mathematics*, 67:73–93, 1996.

[24] Various. "Special section: Credible Computational Fluid Dynamics simulations". *AIAA Journal*, 36(5):665–764, 1998.

[25] P. J. Roache. "Quantification of uncertainty in Computational Fluid Dynamics". In *Annual Review of Fluid Mechanics*, volume 29, pages 123–160, 1997.

[26] D. Thompson, editor. *"The 9th Concise Oxford Dictionary of Current English"*. BCA, by arrangment with Oxford University Press, 1996. ISBN 0-000000-091299.

[27] A. M. Wright and S. R. Turnock. "Techniques for assessing the flow and spatial quality of arbitrary 3D computational meshes". Ship Science Report 108, Department of Ship Science, University of Southampton, 1999.

[28] L. Eça and M. Hoekstra. "On the Numerical Verification of Ship Stern Flow Calculations". In *Technical papers of 1st MARNET CFD Workshop, CIMNE Barcelona*, November 1999.

[29] A. M. Wright and S. R. Turnock. "Convective cell approach for solving incompressible Euler flows: Explicit flux vector splitting scheme using artificial compressibility". Ship Science Report 117, Department of Ship Science, University of Southampton, 1999.

[30] S. R. Turnock. "Interpretation of CFD results for use in ship hyrdodynamic design". In *Proceedings of The International CFD Conference (CFD 99)*, June 1999. Ulsteinvik, Norway.

[31] F. Stern, R. V. Wilson, H. W. Coleman, and E. G. Paterson. "Verification and Validation of CFD Simulations". In *Proceedings of 3rd ASME/JSME Joint Fluids Engineering Conference*, July 1999. Paper no FEDSM99-6913.

[32] I. Demirdzic and M. Perić. "Space conservation law in finite volume calculations of fluid flow". *Int. Journal of Numerical Methods in Fluids*, 8:1037, 1988.

[33] J. F. Thompson. "A Reflection on Grid Generation in the 90s: Trends, Needs and Influences", 1996. http://www.ERC.MsState.Edu/ joe/gridconf/.

[34] R. R. Cosner. "Future Requirements in Surface Modelling and Grid Generation". In *Proceedings of the Surface Modelling, Grid Generation and Related Issues in Computational Fluid Dynamics Workshop*, NASA Conference Publication 3291, page 751, NASA Lewis Research Centre, Cleveland, OH, May 1995.

[35] D. K. Clark, M. D. Salas, and H. A. Haasan. "Euler calculations for multielement airfoils using cartesian grids". *AIAA Journal*, 24:353–359, 1987.

[36] M. Beddhu, M. Y. Jiang, D. L. Whitfield, L. K. Taylor, and A. Arabshahi. "CFD validation of the free surface flow around DTMB Model 5415 using Reynolds Averaged Navier-Stokes Equations". In *Proceedings of Third Osaka colloquium on advanced CFD applications to ship flow and hull form design*, pages 373–393, May 1998.

[37] C. W. Lin, S. Percival, and L. Fisher. "Viscous flow computations on an appended ship by Chimera RANS scheme". In *Proceedings of Third Osaka colloquium on advanced CFD applications to ship flow and hull form design*, pages 161–180, May 1998.

[38] A. Masuko. "Numerical simulation of viscous flow for complex geometries using overlapped grid method". In *Proceedings of Third Osaka colloquium on advanced CFD applications to ship flow and hull form design*, pages 181–197, May 1998.

[39] C. Yang and R. Löhner. "Fully nonlinear ship wave calculation using unstructured grid and parallel computing". In *Proceedings of Third Osaka colloquium*

on advanced CFD applications to ship flow and hull form design, pages 125–150, May 1998.

[40] M. Aftosmis, D. Gaitonde, and T. S. Tavares. "Behaviour of linear reconstruction techniques on unstructured meshes". *AIAA Journal*, 33(11):2038–2049, 1995.

[41] N. P. Weatherill. "Mixed structured-unstructured meshes for aerodynamic flow simulation". *Aeronautical Journal*, 94(934):111–123, 1990.

[42] J. A. Shaw and A. J. Peace. "The modelling of aerodynamic flows by solution of the Euler Equations on mixed polyhedral grids". *International Journal of Numerical Methods in Engineering*, 35:2003–2029, 1992.

[43] D. Sharov and K. Nakahashi. "Hybrid prismatic/tetrahedral grid generation for viscous flow applications". *AIAA Journal*, 36(2):157–162, 1998.

[44] S. Kano and K. Nakahashi. "Flow Computationa Around Delta Wings Using Unstructured Hybrid Grids". *Journal of Aircraft*, 36(2):374–379, March 1999.

[45] R. J. Gomez and E. C. Ma. AIAA paper 94-1859-CP, 1994. Presented at 12th Applied Aerodynamics Conference, Colarado Springs.

[46] E. Turkel. "Accuracy of schemes with non-uniform meshes for compressible fluid-flows". *Applied numerical Mathematics*, 2(6):529–550, 1986.

[47] R. C. Swanson and R. Radespiel. "Cell Centered and Cell Vertex Multigrid Schemes for the Navier-Stokes Equations". *AIAA Journal*, 29(5):697–703, May 1991.

[48] N. C. Rycroft and S. R. Turnock. "Hybrid cell finite volume Euler solutions of flow around a main-jib sail using an IBM SP2". In *Proceedings of Parallel CFD 97, Manchester*, May 1997.

[49] N. C. Rycroft and S. R. Turnock. "3-D Multiblock Grid Generator; FLEXIMESH". Ship Science Report 101, Department of Ship Science, University of Southampton, 1997.

[50] A. M. Wright. "Automated adaptation of spatial grids for flow solutions around marine bodies of complex geometry", 1999. Submitted for transference from the degree of Master of Philosophy to Doctor of Philosophy.

Appendix

Data Structure

```
/* CODE NAME=> eric */
/* FILE celldataA.h */
/* VERSION 1.0 */

/* version control introduced 29/3/99 AMW */
/* This header file contain references to the C library header
   files and also has the basic data structure types  */

#ifndef CELLDATA1_H
#define CELLDATA1_H

typedef struct{
double x, y;
} coord2D;

typedef struct{
double x, y, z;
} coord3D;

typedef struct{
double density;
double u, v, w;
double pressure;
double energy;
} State;


typedef struct{
double mass;
double u_mom;
double v_mom;
double w_mom;
} State_vector;

typedef struct{
State_vector F;
State_vector G;
State_vector H;
} Flux;
```

```
typedef struct{
double mass;
double u_mom, v_mom;
double energy;
} State_vector_2D;

typedef struct{
double mass;
double u_mom, v_mom;
} State_vector_2Da;

#endif

/* CODE NAME=> eric */
/* FILE celldataB.h */
/* VERSION 1.1 */

/* version control introduced 29/3/99 AMW */
/* SECOND LEVEL OF DATA TYPES USED IN CELL GENERATION AND MANIPULATION */

#ifndef CELLDATA2_H
#define CELLDATA2_H

/* The individual members of the data types shall be described and
 their relevance specified

TYPE                   MEMBER                   DESCRIPTION
-----------------------------------------------------------------
Node
int globalID Global ID, to be used in domain construction during
                                    parallel processing
int boundaryID Identifies the boundary condition of the node -
INTERNAL 0
PRESSURE 2
INFLOW 4
OUTFLOW 8
SOLID_WET_FIXED 16
SOLID_WET_MOVE 32
SOLID_DRY_FIXED 64
SOLID_DRY_MOVE 128
FREE_SURFACE 256
PERIODIC 512
```

int linkID Used within free surface code, to link entity to underlying
surface definition

coord coords The nodes placement in physical space
coord velocity The nodes cartesian velocity wrt a stationary point
double volume The volume of the surrounding control volume

NOT ACTIVE -state properties pointer to the flow properties at the node
during the current timestep/iteration
NOT ACTIVE -state_vector residuals pointer to the flow conservative vector
residuals at current iteration/timestep

int noNodes Number of nodes that the node connects to
NodePTR *nodes Array of pointers to adjoining nodes (double pointer)

int on_off For binary tree structure, specifying whither node is
active or not, in current iteration
NodePTR childPTR For binary tree, pointing to children of current node
NodePTR parentPTR For binary tree, pointing to parent of current node

NodePTR nextPTR Pointer to next node in list of current generation of nodes

----

Face
int globalID Global ID, to be used in domain construction during
parallel processing
int boundaryID Identifies the boundary condition of the face - (all
stored in surfdata.h)
INTERNAL 0
PRESSURE 2
INFLOW 4
OUTFLOW 8
SOLID_WET_FIXED 16
SOLID_WET_MOVE 32
SOLID_DRY_FIXED 64
SOLID_DRY_MOVE 128
FREE_SURFACE 256
PERIODIC 512
int noNodes Number of nodes that the face connects to
NodePTR *nodes Array of pointers to vertex nodes (double pointer)

IN CELL CENTRED SCHEME ONLY
Cell int globalID Global ID, to be used in domain construction during paralle

```
processing
int celltype integer defining the type of cell-
celltype = 0 -> 3D hexahedral
celltype = 1 -> 3D tetrahedral
celltype = 2 -> 3D pyramid
celltype = 3 -> 3D prism
celltype = 4 -> 2D trapezoid
coord3D centre The centre of the cell
FacePTR *faces Array of pointers to surrounding faces (double pointer)
NodePTR *nodes Array of pointers to vertex nodes (double pointer)
-----------------------------------------------------------------
 */


/* used for the 2D euler solver */
typedef struct
  {int globalID;
   coord2D coords;
State    properties;
  } NodeB;

/* ------------------------------------------------------     */
/* this is the structured used in the main flow solver itself  */
 struct node
  {int globalID, boundaryID, linkID;
/* =======   */
double volume; /* do we need this? */
double time_step;
   coord3D coords;
coord3D velocity;
/* =======   */
int noNodes;
struct node **joined_to;
/* =======   */
int on_off;
struct node *nextPTR;
struct node *childPTR;
struct node *parentPTR;

  };

typedef struct node Node;
typedef Node *NodePTR;
```

```
/* ----------------------------------------------------- */


/* ----------------------------------------------------- */

#define OPEN_FACE 1
#define CLOSED_FACE 0

class Face{
// friend ;
public:
Face();
void  set_face_ID(int, int);
void     set_face_nodes(int, NodePTR*);
double   area();
double   internal_face_area();
double   external_face_area();
coord3D  centroid();
class Vector   normal();
double   AR();
int      no_subfaces();
int      no_edges();
int      open_or_closed();
NodePTR  base_node();
int      start_node();
int      finish_node();
NodePTR  next_node(int);

/* ======    */
   int globalID, boundaryID;
   int  noNodes;
   NodePTR   *node;
/* =======   */
class Face *nextPTR;
  };

typedef Face *FacePTR;


typedef struct
  {int noNodes;
int total_noNodes;
int noFaces;
int Face_space;
int noConstructNodes;
NodePTR *node;
NodePTR Cnode;
FacePTR *face;
  } N_Grid;

/* -------------------------------------------------------    */
/* -------------------------------------------------------    */
/* -------------------------------------------------------    */
/*  ======= for cell centred schemes =======  */
/* -------------------------------------------------------    */
/* -------------------------------------------------------    */
/* -------------------------------------------------------    */
/* used in the cell centred (hence 'cc') grid */
 struct cc_node
  {int globalID, boundaryID, linkID;
int noNodes;
double volume;
struct cc_node **node;
   coord3D coords;
  };

typedef struct cc_node CC_Node;
typedef CC_Node *CC_NodePTR;

/* -------------------------------------------------------    */
struct cc_face
  {int globalID, boundaryID;
   int  noNodes;
   int  noCells;
   CC_NodePTR   *node;
   struct cell **cell;
  };

typedef struct cc_face CC_Face;
typedef CC_Face *CC_FacePTR;


/* -------------------------------------------------------    */
struct cell
```

```
  {int      globalID, celltype;
int noNodes, noFaces;
CC_NodePTR *node;
   CC_FacePTR  *face;
  };

typedef struct cell Cell;
typedef Cell *CellPTR;

/* ---------------------------------------------------------    */
typedef struct
  {int noNodes;
int noFaces;
int noCells;
Cell *cell;
CC_Face *face;
CC_Node *node;
  } C_Grid;


/* ---------------------------------------------------------    */


#endif

// source code for geometrical class structure manipulation

#include "./universal.h"

Face::Face()
{globalID = LOOSE;
 boundaryID = LOOSE;
 noNodes = 0;
 node = NULL;
}

void Face::set_face_ID(int global, int boundary)
{
   globalID = global;
boundaryID = boundary;
}

double Face::area()
```

```
{double area;

   if (boundaryID == INTERNAL)
area = internal_face_area();
else
area = external_face_area();

return(area);
}

double Face::internal_face_area()
{int i;
 double area, projected_A;
 coord3D base;
 Vector OA, OB;
 Vector vector_A[10];
 Vector face_normal;


// calc the area of each of the faces surrounding the node,
// as well as the normal of the face
for (i = 0; i < no_subfaces(); i ++){

// set up the 2 face edge vectors
OA = Dist_2_point(base_node()->coords, node[3 + i]->coords);
OB = Dist_2_point(base_node()->coords, node[4 + i]->coords);

// calc the area and sub face normal
vector_A[i] = Cross_product(OA, OB);
vector_A[i] = scale_vector(0.5, vector_A[i] );

// and calculate the face normal as a weighted average of the sub-face areas
face_normal = Vect_sum(face_normal, vector_A[i]);
}

face_normal.normalise();

//now set the area, taking only the normal component
area = 0.0;
for (i = 0; i < no_subfaces(); i ++){
projected_A = fabs( Dot_product(face_normal, vector_A[i] ) );

area = area + projected_A;
}

 return(area);
}


double Face::external_face_area()
{int i;
 double area, projected_A;
 Vector OA, OB;
 Vector vector_A[10]; // this was previously dynamically set, but 10
                          // should be more than twice the nessecary
 Vector face_normal;


// calc the area of each of the faces surrounding the node, as well as
   // the normal of the face
for (i = 0; i < no_subfaces(); i ++){

// get the principle sides of the triangular sub-faces
OA = Dist_2_point(base_node()->coords, node[2 + i]->coords);
OB = Dist_2_point(base_node()->coords, node[3 + i]->coords);

// calc the area and sub face normal
vector_A[i] = Cross_product(OA, OB);
vector_A[i] = scale_vector(0.5, vector_A[i] );

// and calculate the face normal as a weighted average of the sub-face areas
face_normal = Vect_sum(face_normal, vector_A[i]);

}

face_normal.normalise();

//now set the area, taking only the normal component
area = 0.0;
for (i = 0; i < no_subfaces(); i ++){
projected_A = fabs( Dot_product(face_normal, vector_A[i] ) );

area = area + projected_A;
}

 return(area);
```

```
}




coord3D Face::centroid()
{int i;
 double area, sub_area[20];
 coord3D centroid, sub_centroid[20];
 Vector OA, OB, temp;


 if (no_subfaces() > 20){
fprintf(stderr,"MEMORY ERROR - more than 20 subfaces; insufficient memory
space to calc face.centroid()\n");
exit(1);
 }

for (i = 0; i < no_subfaces(); i ++){

  // get the principle sides of the triangular sub-faces
  if (boundaryID == INTERNAL){
 OA = Dist_2_point(base_node()->coords, node[3 + i]->coords);
 OB = Dist_2_point(base_node()->coords, node[4 + i]->coords);
 sub_centroid[i].x = (1.0/3.0)*(base_node()->coords.x
                                  + node[3+i]->coords.x
                                  + node[4+i]->coords.x);
 sub_centroid[i].y = (1.0/3.0)*(base_node()->coords.y
                                  + node[3+i]->coords.y
                                  + node[4+i]->coords.y);
 sub_centroid[i].z = (1.0/3.0)*(base_node()->coords.z
                                  + node[3+i]->coords.z
                                  + node[4+i]->coords.z);
  }
  else{
 OA = Dist_2_point(base_node()->coords, node[2 + i]->coords);
 OB = Dist_2_point(base_node()->coords, node[3 + i]->coords);
 sub_centroid[i].x = (1.0/3.0)*(base_node()->coords.x
                                  + node[2+i]->coords.x
                                  + node[3+i]->coords.x);
 sub_centroid[i].y = (1.0/3.0)*(base_node()->coords.y
                                                  + node[2+i]->coords.y
                                                  + node[3+i]->coords.y);
 sub_centroid[i].z = (1.0/3.0)*(base_node()->coords.z
                                                  + node[2+i]->coords.z
                                                  + node[3+i]->coords.z);

  }

  // calc the area
temp = Cross_product(OA, OB);
temp = scale_vector(0.5, temp );
sub_area[i] = temp.magnitude();
}

area = 0.0;
centroid.x = centroid.y = centroid.z = 0.0;

for (i = 0; i < no_subfaces(); i ++){

area += sub_area[i];

centroid.x += (sub_area[i]*sub_centroid[i].x);
centroid.y += (sub_area[i]*sub_centroid[i].y);
centroid.z += (sub_area[i]*sub_centroid[i].z);
}

centroid.x /= area;
centroid.y /= area;
centroid.z /= area;

return(centroid);
}




Vector Face::normal()
{int i, start;
 coord3D base;
 Vector the_normal;
 Vector OA, OB;
 Vector vector_A;


the_normal.set_vector(0.0, 0.0, 0.0);
if (boundaryID == INTERNAL){
base = node[2]->coords;
start = 4;
}
else{
base = node[0]->coords;
start = 3;
}

// calc the area of each of the faces surrounding the node, as well as the
   // normal of the face

for (i = 0; i < no_subfaces(); i ++){

// get the principle sides of the triangular sub-faces
OA = Dist_2_point(base, node[start - 1 + i]->coords);
OB = Dist_2_point(base, node[start + i]->coords);

// the cross product gives a vector of twice the area in the
      // direction of the normal
vector_A = Cross_product(OA, OB);

// these area weighted normals can be summed and normalised
the_normal = Vect_sum(the_normal, vector_A);
}

the_normal.normalise();

return(the_normal);
}

/* Define the face aspect ratio (AR) as the average of the
   aspect ratios of all the subfaces.              */
/*   The subface aspect ratio is taken to be the area of the
   subface divided by half the chord squared    */
/*   ie AR = A / 0.5c^2     from A = 0.5sc  and AR = s/c            */

double Face::AR()
{int i;
```

```
 double AR, subface_AR, subface_area;
 Vector OA, OB, AB;

   AR = 0.0;

for (i = 0; i < no_subfaces(); i ++){

  if (boundaryID == INTERNAL){
 OA = Dist_2_point(base_node()->coords, node[3 + i]->coords);
 OB = Dist_2_point(base_node()->coords, node[4 + i]->coords);
 AB = Dist_2_point(node[3 + i]->coords, node[4 + i]->coords);
  }
  else{
 OA = Dist_2_point(base_node()->coords, node[2 + i]->coords);
 OB = Dist_2_point(base_node()->coords, node[3 + i]->coords);
 AB = Dist_2_point(node[2 + i]->coords, node[3 + i]->coords);
  }

subface_area = 0.5*(Cross_product(OA, OB).magnitude());

subface_AR = subface_area/( 0.5*SQR(AB.magnitude()) );

if (subface_AR < 1)
  subface_AR = 1.0/subface_AR;

AR += subface_AR;
}

AR *= 1.0/no_subfaces();

 return(AR);
}



int Face::open_or_closed()
{int answer;

   if (INTERNAL == boundaryID){
  if (node[3]->globalID == node[ noNodes-1 ]->globalID)
 answer = CLOSED_FACE;
  else
 answer = OPEN_FACE;
```

```
}
else{
  if (node[2]->globalID == node[ noNodes-1 ]->globalID)
 answer = CLOSED_FACE;
  else
 answer = OPEN_FACE;
}
 return(answer);
}



NodePTR Face::base_node()
{NodePTR answer;

  if (boundaryID == INTERNAL)
 answer = node[2];
  else
 answer = node[0];

  return(answer);
}


int Face::no_subfaces()
{int answer;

  if (boundaryID == INTERNAL)
 answer = noNodes - 4;
  else
 answer = noNodes - 3;

  return(answer);
}


// this routine gives the number of external edges on the face
int Face::no_edges()
{int answer;

if (INTERNAL == boundaryID){

if (node[3]->globalID == node[ noNodes-1 ]->globalID) // enclosed face
```

```
answer = noNodes - 4;
else
answer = noNodes - 2;
} // end of internal face
else{
if (node[2]->globalID == node[ noNodes-1 ]->globalID)
answer = noNodes - 3;
else
answer = noNodes - 1;
}
 return(answer);
}


// the first node on the external edge of the face - not necessarily
// the first construction node
int Face::start_node()
{int start;

if (INTERNAL == boundaryID){
if (node[3]->globalID == node[ noNodes-1 ]->globalID) // enclosed face
start = 3;
else
start = 2;
} // end of internal face
else{
      if (node[2]->globalID == node[ noNodes-1 ]->globalID) // enclosed face
start = 2;
else
start = 0;
}
 return(start);
}

// and the last node defining the extents of the face. If the face
// is enclosed, then the repeated node is referenced at START
int Face::finish_node()
{int finish;

if (INTERNAL == boundaryID){
if (node[3]->globalID == node[ noNodes-1 ]->globalID){ // enclosed face
finish = noNodes - 1;
}
```

```
else{
finish = noNodes;
}
} // end of internal face
else{
if (node[2]->globalID == node[ noNodes-1 ]->globalID){
finish = noNodes - 1;
}
else{
finish = noNodes;
}
}
return(finish);
}

// Sends back the next node on the edge of the face. Not a great
// piece of coding; there are possibilities for a mess passing thu'
NodePTR Face::next_node(int current)
{int X;
 NodePTR answer;

 X = current + 1;

 if (current == noNodes-1){
if (INTERNAL == boundaryID){
  if (node[3] == node[ noNodes-1 ]) // enclosed face
 X = 4;
  else
 X = 2;
}
else{ // boundary face
  if (node[2] == node[ noNodes-1 ]) // enclosed face
 X = 3;
  else
 X = 0;
}
 }

 if (current == 0){
if (boundaryID != INTERNAL)
  X = 2;
else
  exit(2);
  }

  answer = node[X];

 return(answer);
 }
```