FLITE SYSTEM

Version 4

Theoretical Manual

O. Hassan , **K. Morgan** and **N.P. Weatherill**

Swansea University
Singleton Park
Swansea SA2 8PP, U. K.

June 2, 2008

Contents

Chapter 1

Geometry Modelling

1.1 Introduction

The boundary of the domain to be discretised needs to be represented in a suitable manner before the generation procedure can start. If the automatic discretisation of an arbitrary domain is to be achieved, the mathematical description of the domain topology ought to possess the greatest possible generality. The computer implementation of this description must provide means for automatically computing any geometrical quantity relevant to the generation procedure. Solid modelling provides [81] the most general up-to-date set of methods for the computational representation and analysis of general shapes matching the above requirements.

In this section we give a brief description of the geometry modelling strategy that we employ. More sophisticated representations that ease the task of performing quick geometry modifications could also be used [19].

1.2 Description of the computational domain

In the planar two dimensional case, the boundary is represented by closed loops of composite cubic spline curves [19]. In three dimensions, and following solid modelling ideas, the domain to be discretised is viewed as a region in $\mathbb{R}^3$ bounded by a general polyhedron whose faces are regions on curved surfaces which intersect along curves. The edges of the polyhedron are segments on these intersection curves. In our notation, the portions of these curves and surfaces needed to define the boundary of the three-dimensional domain of interest are called *curve* and *surface components* respectively. A surface component is represented as a region —a patch— on a surface delimited by curve components. Each curve component is common to two surface components and is a segment of the intersection curve between their respective support surfaces. The approximate representation of the curves and surfaces where boundary components lay is accomplished by means of composite curves and surfaces [19].

1.3 Curve Representation

The parametric definition of a curve consists of a piecewise interpolation of cubic polynomials through an ordered set of data points. The order in which these points are given defines the orientation. In the Ferguson representation [22], each cubic polynomial is expressed, in terms of the position and tangent vectors at the two end points, as

$$\mathbf{r}(v) = \begin{bmatrix} 1 & v & v^2 & v^3 \end{bmatrix} \mathbf{C} \begin{bmatrix} \mathbf{r}^{(1)} \\ \mathbf{r}^{(2)} \\ \mathbf{t}^{(1)} \\ \mathbf{t}^{(2)} \end{bmatrix} \quad 0 \leq v \leq 1 \tag{1.1}$$

where $\mathbf{r}$ denotes the position of a point respect to a cartesian frame of reference in $\mathbb{R}^3$, $\mathbf{r}^{(1)}$ and $\mathbf{r}^{(2)}$ represent the coordinates of the end points of the segment, $\mathbf{t}^{(1)}$ and $\mathbf{t}^{(2)}$ are their respective tangents and $\mathbf{C}$ is a constant matrix given by

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \tag{1.2}$$

The tangent to the curve is computed according to

$$\mathbf{t}(v) = \mathbf{r}'(v) = \frac{d\mathbf{r}(v)}{dv} \tag{1.3}$$

The number of data points, and their spatial distribution, should be given in such a manner that the interpolated curve accurately approximates the intersection of the corresponding surface components. The interpolation problem consists of fitting a parametric spline, defined in a piecewise manner, through a set of n points $\mathbf{r}_j$; $j = 1, \ldots, n$. At interior points, continuity of slopes is guaranteed for any choice of the tangent vectors, provided that a unique tangent vector is used for the definition of the two adjacent cubic segments. However, by employing a simple procedure [19], these vectors can be determined so that continuity of curvature is achieved throughout the interpolated curve. At the two end points, *zero curvature* is assumed. Note that the expressions given above are valid in two and three dimensions. The only difference in the two cases being the number of components of the vectors $\mathbf{r}$ and $\mathbf{t}$.

1.4 Surface Representation

The mathematical representation of a surface is obtained by interpolating a composite surface, made up of quadrilateral patches, through a topologically rectangular set of data points $\mathbf{r}_{jk}$; $j = 1, \ldots, m$; $k = 1, \ldots, n$. Two families of parametric lines are obtained by interpolating spline curves, first through the points of constant j and then through the points of constant k. The procedure used for interpolating each spline curve is that described in the previous section. The mathematical expression for every quadrilateral surface patch is given, in terms

of the four cubic curves that form its boundary and the twist vector at the four corner points, as

$$\mathbf{r}(v,w) = \left[\begin{array}{cccc} 1 & v & v^2 & v^3 \end{array}\right] \mathbf{C} \left[\begin{array}{cccc} \mathbf{r}^{(1)} & \mathbf{r}^{(4)} & \mathbf{r}_{,w}^{(1)} & \mathbf{r}_{,w}^{(4)} \\ \mathbf{r}^{(2)} & \mathbf{r}^{(3)} & \mathbf{r}_{,w}^{(2)} & \mathbf{r}_{,w}^{(3)} \\ \mathbf{r}_{,v}^{(1)} & \mathbf{r}_{,v}^{(4)} & \mathbf{r}_{,vw}^{(1)} & \mathbf{r}_{,vw}^{(4)} \\ \mathbf{r}_{,v}^{(2)} & \mathbf{r}_{,v}^{(3)} & \mathbf{r}_{,vw}^{(2)} & \mathbf{r}_{,vw}^{(3)} \end{array}\right] \left[\begin{array}{c} 1 \\ w \\ w^2 \\ w^3 \end{array}\right] \quad 0 \leq v,\, w \leq 1 \tag{1.4}$$

where $\mathbf{C}$ is the matrix previously defined in 1.2, $\mathbf{C}^T$ is its transpose, and the comma denotes partial differentiation, i. e.,

$$\mathbf{r}_{,v} = \frac{\partial \mathbf{r}}{\partial v}\,; \quad \mathbf{r}_{,w} = \frac{\partial \mathbf{r}}{\partial w}\,; \quad \mathbf{r}_{,vw} = \frac{\partial \mathbf{r}}{\partial v \partial w} \tag{1.5}$$

Here the notation employed to denote the corner points of the patch is

$$\mathbf{r}^{(1)} = \mathbf{r}(0,0)\,; \quad \mathbf{r}^{(2)} = \mathbf{r}(1,0)\,; \quad \mathbf{r}^{(3)} = \mathbf{r}(1,1)\,; \quad \mathbf{r}^{(4)} = \mathbf{r}(0,1) \tag{1.6}$$

This representation uses a Hermite interpolation between opposite boundaries of the patch [13]. The twist vectors $(\mathbf{r}_{,vw})_{jk}$ at the corner points are computed so that overall second order continuity is achieved on the interpolated surface. The implementation details of this algorithm can be found in [74]. For convenience, global parametric coordinates u_1 and u_2 are defined. For the patch (j,k) these coordinates are related to the local patch coordinates v and w according to $u_1 = v+j-1$ and $u_2 = w+k-1$. In this way, a global mapping $\mathbf{r}(u_1,u_2)$ is established between the rectangular region in the parametric plane defined by $0 \leq u_1 \leq p\,$; $0 \leq u_2 \leq q$ and the tensor product surface. The orientation of the surface is defined by specifying the outward normal $\mathbf{n}$ which points towards the region to be discretised. This normal $\mathbf{n}$ is given by

$$\mathbf{n} \;=\; \mathbf{r}_{,u} \times \mathbf{r}_{,v} \tag{1.7}$$

where $\times$ denotes the vector product.

Chapter 2

Mesh Generation

2.1 Introduction

The most widely used approach for solving the problem of producing a discretisation of a computational domain of complex geometrical shape is the use of the multi-block method of mesh generation. In this method, the domain is initially subdivided into an unstructured assembly of hexahedral blocks and a structured hexahedral mesh is employed within each block. Flow simulations may then be made by using an appropriately modified structured grid based flow solver. The adequacy of the initial blocking of the domain is frequently crucial to the quality of the final mesh which is produced and, even though many impressive demonstrations have been made which indicate the viability of the approach (see, for example, [1, 38]) the analyst frequently complains about the lead time necessary to accomplish this task.

The use of an unstructured mesh of tetrahedra is an alternative solution to this problem and appears to offer the possibility of automating the complete procedure, so that mesh generation times can be significantly reduced. In an unstructured mesh, the number of points and elements which are neighbours to an interior point is not kept constant throughout the domain. This lack of regularity in the mesh means that a unstructured flow algorithm represents an additional cost of computer time and memory when compared with its structured counterparts. On the other hand, it offers, as a counter balance, a greater versatility and geometrical flexibility to the mesh generating process.

In order to take full advantage of these characteristics, the mesh generation procedure ought to comply with the following requirements:

- The algorithm should be able to handle arbitrary geometries in a fully automatic manner and with minimum user intervention.
- The input data should be reduced to a computerized geometric representation of the domain to be discretized.
- The scheme should provide control over the spatial variation of element size and shape through the domain.
- Adaptive methods should be incorporated in the process aiming to produce

the most accurate approximation of the solution for a given number of points.

The mesh generator employed in the system has been developed to match these requirements. The algorithmic procedure for the generation of elements and nodes is three-dimensional extension of the triangulation method proposed in [39]. This is based upon a generalization of the advancing front technique.

2.2 Characterisation of the mesh: Mesh parameters

The geometrical characteristics of a general mesh are locally defined in terms of certain mesh parameters. If N, is the number of dimensions (2 or 3) then, the parameters used are a set of N mutually orthogonal directions $\boldsymbol{\alpha}_i$; $i = 1, \ldots, N$, and N associated element sizes δ_i; $i = 1, \ldots, N$. Thus, at a certain point, if all N element sizes are equal, the mesh in the vicinity of that point will consist of approximately equilateral elements. To aid the mesh generation procedure, a transformation $\mathbf{T}$ which is a function of $\boldsymbol{\alpha}_i$ and δ_i is defined. This transformation is represented by a symmetric $N \times N$ matrix and maps the physical space onto a space in which elements, in the neighbourhood of the point being considered, will be approximately equilateral with unit average size. This new space will be referred to as the normalised space. For a general mesh this transformation will be a function of position. The transformation $\mathbf{T}$ is the result of superimposing N scaling operations with factors $\frac{1}{\delta_i}$ in each $\boldsymbol{\alpha}_i$ direction. Thus

$$\mathbf{T}(\boldsymbol{\alpha}_i, \delta_i) = \sum_{i=1}^{N} \frac{1}{\delta_i} \boldsymbol{\alpha}_i \otimes \boldsymbol{\alpha}_i \tag{2.1}$$

where $\otimes$ denotes the tensor product of two vectors.

2.3 Mesh control

The inclusion of adequate mesh control is a key ingredient in ensuring the generation of a mesh of the desired form. Control over the characteristics is obtained by the specification of a spatial distribution of mesh parameters. This is accomplished by means of the *background mesh* and the *sources distribution*.

2.3.1 Background mesh

The background mesh is used for interpolation purposes only and is made up of triangles in two dimensions and tetrahedra in three dimensions. Values of $\boldsymbol{\alpha}_i$ and δ_i are defined at the nodes of the background mesh. The background mesh employed must cover the region to be discretised (see Figure 2.1). In the generation of an initial mesh for the analysis of a particular problem, the background mesh will usually consist of a small number of elements. The generation of the background mesh can in this case be accomplished without resorting to sophisticated procedures e. g. a background mesh consisting of a single element can be used to impose the requirement of linear or constant spacing and stretching through the computational domain.

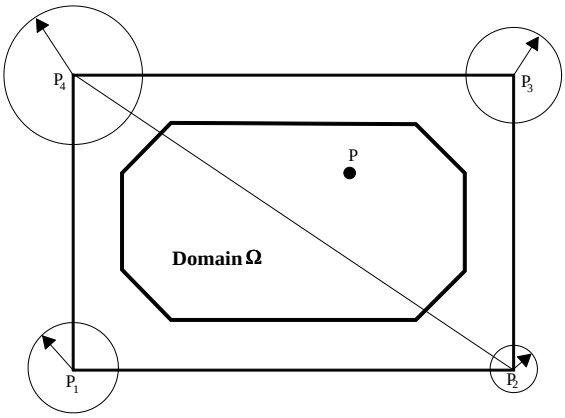

Figure 2.1: The background mesh used to control mesh point spacing

2.3.2 Sources distribution

For complex geometries, the definition by hand of a background mesh could become a very cumbersome task. The use of a sources distribution eases the specification of the mesh parameters at specific regions in the computational domain like, for instance, leading and trailing edges of wings. In this method an isotropic spatial distribution of element sizes is specified as a function of the distance x from the point of interest to a 'source': a point, line segment or triangle. The form of the function chosen in this work is

$$\delta(x) = \begin{cases} \delta_1 & \text{if } x \leq x_c \\ \delta_1 \, e^{|\frac{x-x_c}{D-x_c}| \log 2} & \text{if } x \geq x_c \end{cases} \tag{2.2}$$

This function is local in character and allows for a rapid increase in element size thus keeping the number of generated elements around the source within reasonable bounds. δ_1, D, and x_c are user-specified values that control the shape of $\delta(x)$. Figure 2.2 shows, in schematic form, three basic types of sources. Although there are many variants of the definition of a source now in the literature, the fundamental features of a point source, as already described , are defined by A position, Q, within the domain. At Q, the required mesh point spacing is defined, δ. A circle is specified of radius r_1, within which the user specified mesh point density, δ, is defined. A second circle is specified of radius r_2, where, $r_2 > r_1$. Within the region defined between the circle radius r_1 and the circle radius r_2, the point spacing will decay from δ to that specified by the background mesh. Hence, for a point source with the structure just defined, the user must specify 4 parameters. However, this does not involve the intricacies of a mesh connectivity as required with a background mesh. In fact, the background mesh that accompanies sources is effectively redundant, since uniform spacing everywhere can be the default condition. Hence, no interpolation is required.

The extension to a line source, a triangle source, or even a volume source is straightforward. Figure 2.2 shows, in schematic form, a line and triangle source.

These two allow the user to easily specify the required mesh point density over regions of the domain.

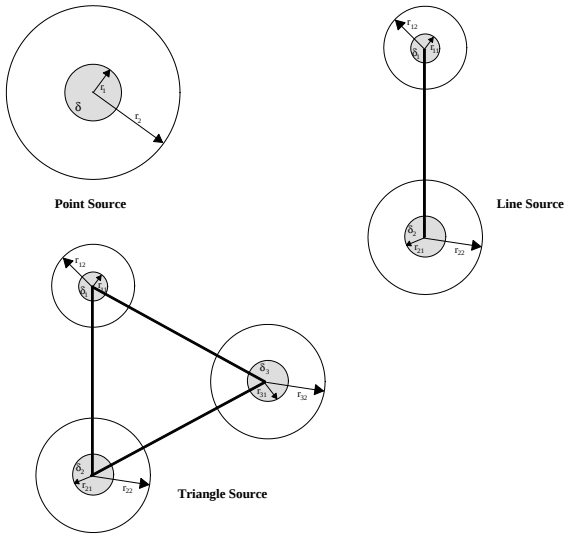

Figure 2.2: Point, Line and Triangular sources

2.3.3 Calculation of the transformation T

The generation process is always carried out in the normalised space. The transformation $\mathbf{T}$ given by 2.1 is repeatedly used to transform regions in the physical space into regions in the normalised space. In this way the process is greatly simplified, as the desired size for a side, triangle or tetrahedra in this space is always unity. After the element has been generated, the coordinates of the newly created point, if any, are transformed back to the physical space using the inverse transformation.

At any point of the computational domain the transformation $\mathbf{T}$ is computed as follows. First, the element of the background mesh that contains the point is found and the transformation $\mathbf{T}^b$ is computed by linearly interpolating its components from the element nodal values. The stretching directions $\boldsymbol{\alpha}_i^b$ and corresponding spacings δ_i^b; $i = 1, 2, 3$ are obtained from the eigenvalues and eigenvectors of the matrix $\mathbf{T}^b$. The the spacings δ_i^b are then modified to account for the source distribution. The new spacings δ_i^* at the point are computed as the smallest of the spacings defined by all the sources and the current spacing δ_i^b. Finally, the transformation $\mathbf{T}$ is obtained by subtituting the values $\boldsymbol{\alpha}_i^b$ and δ_i^* in the formula 2.1.

2.4 Curve discretisation

The discretisation of the boundary curve components is achieved by positioning nodes along the curve according to a spacing dictated by the local value of the mesh parameters. Consecutive points are joined by straight lines to form sides.

In order to determine the position and number of nodes to be created on each curve component, the following steps are followed:

i) Subdivide recursively each cubic segment into smaller cubic segments until their length is smaller than a certain prescribed value. A safe choice for this value is the minimum spacing specified in the background mesh but often, considerably larger values can be taken. The length of each cubic segment is computed numerically. When subdividing a cubic segment, the position and tangent vectors corresponding to the new data points can be found directly from the original definition of the segment.

ii) For all the data points $\hat{\mathbf{r}}_j$; $j = 1, \ldots, n$ (i. e. those used to define the curve and those created to satisfy the maximum length criterion), interpolate from the background mesh the coefficients of the transformation $\mathbf{T}_j$ and transform the position and tangent vectors i. e. $\hat{\mathbf{r}}_j = \mathbf{T}_j\, \mathbf{r}_j$ and $\hat{\mathbf{t}}_j = \mathbf{T}_j\, \mathbf{t}_j$ The new position and tangent vectors $\hat{\mathbf{r}}_j$ and $\hat{\mathbf{t}}_j$; $j = 1, \ldots, n$ define a spline curve which can be interpreted as the image of the original curve component in the normalised space. It must be noted that because of the approximate nature of this procedure, the new curve will in general have discontinuities of curvature even though the curvature of the original curve varies continuously.

iii) Compute the length of the curve in the normalised space and subdivide it into segments of approximately unit length. For each newly created point, calculate the cubic segment in which it is contained and its parametric coordinate. This information is used to determine the coordinates of the new nodes in the physical space, using the curve component definition.

2.5 Surface discretisation

The method followed for the triangulation of the surface components is an extension of the mesh generation procedure for planar domains described above. The discretisation of each surface component is accomplished by generating a two-dimensional mesh of triangles in the parametric plane (u_1, u_2) and then using the mapping $\mathbf{r}(u_1, u_2)$ defined in the previous chapter. This mapping establishes a one-to-one correspondence between the surface component and a region on the parametric plane (u_1, u_2). Thus, a consistent triangular mesh in the parametric plane will be transformed, by the mapping $\mathbf{r}(u_1, u_2)$, into a valid triangulation of the surface component. The construction of the triangular mesh in the parameter plane (u_1, u_2) using the two dimensional mesh generator, requires the determination of an appropriate spatial distribution of the two dimensional mesh parameters. These consist of a set of two mutually orthogonal directions $\boldsymbol{\alpha}_i^\star$; $i = 1, 2$, and two associated element sizes $\delta_i^\star$; $i = 1, 2$.

The two dimensional mesh parameters in the (u_1, u_2) plane can be evaluated from the spatial distribution of the three dimensional mesh parameters and the distortion and stretching introduced by the mapping. To illustrate this process, consider a point $P^\star$ in the parametric plane of coordinates $(u_1^\star, u_2^\star)$ where the values of the mesh parameters $\boldsymbol{\alpha}_i^\star, \delta_i^\star$; $i = 1, 2$ are to be computed. Its image

on the surface will be the point $P = \mathbf{r}(u_1^\star, u_2^\star)$. The transformation between the physical space and the normalised space at this point $\mathbf{T}^P$ can be obtained by direct interpolation from the background mesh. A new mapping, valid in the neighbourhood of point P, can now be defined between the parametric plane (u_1, u_2) and the normalised space as

$$\mathbf{R}(u_1, u_2) = \mathbf{T}_P \, \mathbf{r}(u_1, u_2) \tag{2.3}$$

A curve in the parametric plane passing through point $P^\star$ and with unit tangent vector $\boldsymbol{\beta} = (\beta_1, \beta_2)$ at this point, is transformed by the above mapping into a curve in the normalised space passing through the point $\mathbf{T}^P P$. The arc length parameters $ds^\star$ and ds, along the original and transformed curves respectively, are related by the expression [**?**]

$$ds^2 = \left\{ \sum_{i,j=1}^{2} \frac{\partial \mathbf{R}}{\partial u_i} \frac{\partial \mathbf{R}}{\partial u_j} \beta_i \beta_j \right\} ds^{\star 2} \tag{2.4}$$

Assuming that this relation between the arc length parameters also holds for the spacings, we can compute the spacing δ_β along the direction $\boldsymbol{\beta}$ in the parameter plane as

$$\frac{1}{\delta_\beta} = \sqrt{\sum_{i,j=1}^{2} \frac{\partial \mathbf{R}}{\partial u_i} \frac{\partial \mathbf{R}}{\partial u_j} \beta_i \beta_j} \tag{2.5}$$

The two-dimensional mesh parameters $\boldsymbol{\alpha}_i^\star$, $\delta_i^\star$; $i = 1, 2$ are determined from the directions in which δ_β attains an extremum. This reduces to finding the eigenvalues and eigenvectors of a symmetric 2×2 matrix.

To form the initial front, the (u_1, u_2) coordinates of the nodes already generated on the boundary curve components have to be computed. As the mapping $\mathbf{r}(u_1, u_2)$ cannot be inverted analytically, the coordinates (u_1, u_2) of such points are found numerically by using a direct iteration procedure [74].

2.5.1 Triangle generation in two-dimensional domains

The triangle generation algorithm utilises the concept of a generation front [26, 53, 75]. At the start of the process the front consists of the sequence of straight line segments which connect consecutive boundary nodes. During the generation process, any straight line segment which is available to form an element side is termed active, whereas any segment which is no longer active is removed from the front. Thus while the domain boundary will remain unchanged, the generation front changes continuously and needs to be updated whenever a new element is formed.

In the process of generating a new triangle the following steps are involved:

i) Select a side AB of the front to be used as a base for the triangle to be generated. Here, the criterion is to choose the shortest side. This is especially advantageous when generating irregular meshes.

ii) Interpolate from the background mesh the transformation $\mathbf{T}$ at the centre of the side M and apply it to the nodes in the front which are relevant to the triangulation. In our implementation we define the relevant points to be all those which lie inside the circle of centre M and radius three times the length of the side being considered. Let $\hat{A}$, $\hat{B}$ and $\hat{M}$ denote the positions in the normalised space of the points A, B and M respectively.

iii) Determine, in the normalised space, the ideal position $\hat{P}_1$ for the vertex of the triangular element. The point $\hat{P}_1$ is located on the line perpendicular to the side that passes through the point $\hat{M}$ and at a distance δ_1 from the points $\hat{A}$ and $\hat{B}$. The direction in which $\hat{P}_1$ is generated is determined by the orientation of the side. The value δ_1 is chosen according to:

$$\delta_1 = \begin{cases} 1.00 & \text{if } 0.55 \times L < 1.00 < 2.00 \times L \\ 0.55 \times L & \text{if } 0.55 \times L < 1.00 \\ 2.00 \times L & \text{if } 1.00 > 2.00 \times L \end{cases} \tag{2.6}$$

where L is the distance between points $\hat{A}$ and $\hat{B}$ Only in situations where the side AB happens to have characteristics very different from those specified by the background mesh will the value of δ_1 be different from unity. However, the above inequalities must be taken into account to ensure geometrical compatibility. Expression (2.6) is purely empirical and different inequalities could be devised to serve the same purpose.

iv) Select other possible candidates for the vertex and order them in a list. Two types of points are considered viz. (a) all the nodes $\hat{Q}^1, \hat{Q}^2, \ldots$ in the current generation front which are, in the normalised space, interior to a circle with centre $\hat{P}^1$ and radius $r = \delta_1$, and (b) the set of points $\hat{P}^1, \ldots, \hat{P}^5$ generated along the height $\hat{P}^1\hat{M}$. For each point $\hat{Q}^i$, construct the circle with centre $\hat{C}^i_Q$, on the line defined by points $\hat{P}^1$ and $\hat{M}$ and which passes through the points $\hat{Q}^i$, $\hat{A}$ and $\hat{B}^1$. The position of the centres $\hat{C}^i_Q$, of these circles on the line $\hat{P}^1\hat{M}$ defines an ordering of the $\hat{Q}^i$ points. A list is created that contains all the $\hat{Q}^i$ points in which the point with the furthest centre from $\hat{P}^1$ in the direction $\hat{P}^1\hat{M}$ appears at the head of list. The points $\hat{P}^1, \ldots, \hat{P}^5$ are added at the end of this list.

v) Select the best connecting point. This is the first point in the ordered list which gives a consistent triangle. Consistency is guaranteed by ensuring that none of the newly created sides intersects with any of the existing sides in the front.

vi) Finally, if a new node is created, its coordinates in the physical space are obtained by using the inverse transformation $\mathbf{T}^{-1}$.

vii) Store the new triangle and update the front by adding/removing the relevant sides.

2.5.2 Mesh quality enhancement

In order to enhance the quality of the generated mesh, two post-processing procedures are applied. These procedures, which are local in nature, do not alter the total number of points or elements in the mesh.

- *Diagonal swapping*: This changes the connectivities among nodes in the mesh without altering their position. This process requires a loop over all the element sides excluding those sides on the boundary. For each side AB (Figure **??**) common to the triangles ABC and ADB one considers the possibility of swapping AB by CD, thus replacing the two triangles ABC and ADB by the triangles ADC and BCD. The swapping is performed if a prescribed regularity criterion is satisfied better by the new configuration than by the existing one. In our implementation, the swapping operation is performed if the minimum angle occuring in the new configuration is larger than in the original one.

- *Mesh smoothing*: This alters the positions of the interior nodes without changing the topology of the mesh. The element sides are considered as springs of stiffness proportional to the length of the side. The nodes are moved until the spring system is in equilibrium. The equilibrium positions are found by iteration. Each iteration amounts to performing a loop over the interior points and moving their coordinates to coincide with those of the centroid of the neighbouring points. Usually three to five iterations are performed.

The combined application of these two post-processing algorithms is found to be very effective in improving the smoothness and regularity of the generated meshes.

2.6 Volume Mesh Generation

2.6.1 The Delaunay Construction

Dirichlet in 1850 proposed a method whereby a given domain could be systematically decomposed into a set of packed convex polygons. This basic construction will be described.

Firstly, consider a particularly simple construction. Given two points in the plane, P_1 and P_2, the perpendicular bisector of the line joining the two points subdivides the plane into two regions, V_1 and V_2. The region V_1 is the space closer to P_1 than to P_2 and the region V_2 is the space closer to P_2 than to P_1 Figure 2.3.

Extending these ideas, it is clear that for a given set of points in the plane $P_i, i = 1, N$, the regions V_i are territories which can be assigned to each point such that V_i represents the space closer to P_i than to any other point in the set.

This geometrical construction of tiles is known as the Dirichlet tessellation. This tessellation of a closed domain results in a set of non-overlapping convex polygons, called Voronoi regions, covering the entire domain. The boundaries of

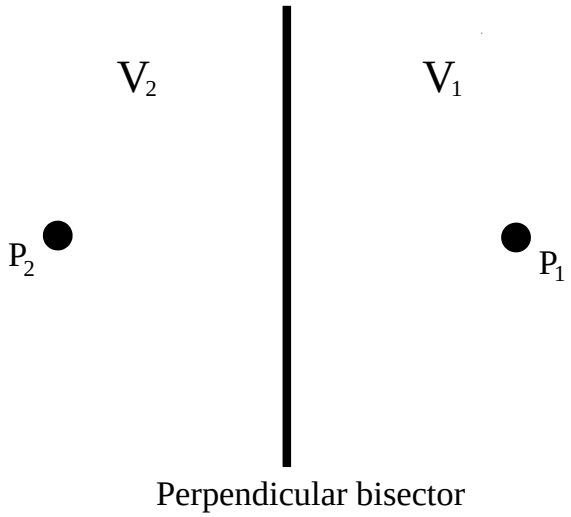

Figure 2.3: Voronoi regions associated with two points

the Voronoi regions form the Voronoi diagram. The intersections of the boundaries of the Voronoi regions are called Voronoi vertices Figure 2.4.

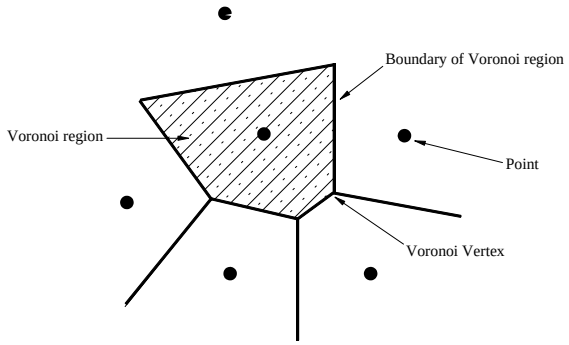

Figure 2.4: Voronoi diagram or Dirichlet Tessellation

A more formal definition can be stated. If a set of points is denoted by $P_i, i = 1, N$ then the Voronoi region V_i can be defined as

$$V_i = P : ||P - P_i|| < ||P - P_j|| \qquad for \quad all \quad j \neq i \tag{2.7}$$

i.e. the Voronoi region V_i is the set of all points that are closer to P_i than to any other point. The sum of all points forms a Voronoi polygon.

If all point pairs, which have some segment of the Voronoi boundary in common, are joined by straight lines, the result is a triangulation of the convex hull of the set of points P_i. This triangulation is known as the Delaunay triangulation. An example of this construction, illustrated in two dimensions, is shown in Figure 2.5

From this definition of the Voronoi diagram, it is apparent that in two dimensions, the territorial boundary which forms a side of a Voronoi polygon must be

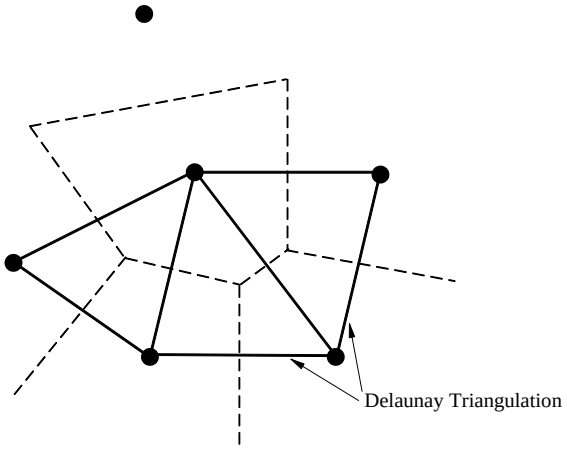

Figure 2.5: The Delaunay and Voronoi constructions

midway between the two points that it separates, and is thus a segment of the perpendicular bisector of the line joining these two points. In Figure 2.6, the line segment V_1V_3 is the perpendicular bisector between the points P_1 and P_3. The point Q is, therefore, the mid point of the line P_1P_3 and the line segment P_1P_3 is perpendicular to the line segment V_1V_3. It follows that the Delaunay triangulation and the Voronoi Diagram are dual constructions that are orthogonal to each other segment V_1V_3. It follows that the Delaunay triangulation and the Voronoi Diagram are dual constructions that are orthogonal to each other

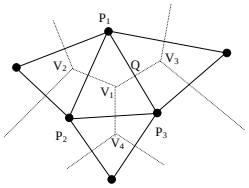

Figure 2.6: Line segments between the Voronoi diagram and Delaunay triangles

It follows that the Voronoi Vertex V_1 must be equidistant between points P_1 and P_3, since V_1 lies on the perpendicular bisector of P_1 and P_3. However, in a similar way, V_1 must be equidistant between the points P_1 and P_2, since V_1 also lies on the perpendicular bisector of P_1 and P_2. If V_1 is equidistant from P_1 and P_3 and also P_2, then the Voronoi vertex V_1 is at the centre of a circle which can be constructed to pass through the points P_1, P_2 and P_3 Figure 2.7. segment V_1V_3. It follows that the Delaunay triangulation and the Voronoi Diagram are dual constructions that are orthogonal to each other

Equivalent constructions can be defined in higher dimensions. In three dimensions, the territorial boundary which forms a face of a Voronoi polyhedron is equidistant between the two points which it separates. If all point pairs which have a common face in the Voronoi construction are connected then a set of tetrahedra is formed which covers the convex hull of the data points. Further details

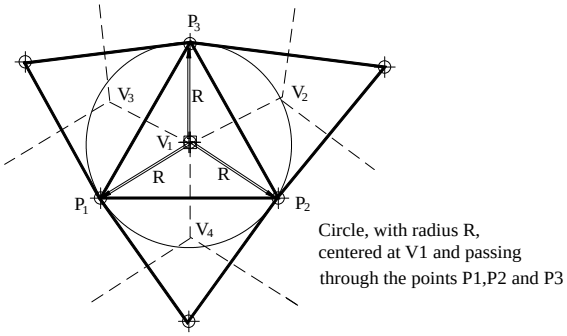

Figure 2.7: Circumscribed circle

will be given in Section 8.7.

2.6.2 Properties of the Delaunay Triangulation

The In-Circle Criterion

The Delaunay triangulation has some rather interesting properties. One of particular interest is the so-called in-circle criterion. The vertices of the Voronoi diagram are at the circumcentres of the circles which pass through the three points which forms a triangle. It follows from the definition of the Dirichlet tessellation that no points, other than the so-called forming points which form the triangles fall within the circles. If a point did fall inside then this would contradict the basic definition. This geometrical property is the in-circle criterion.

This interesting property can be used to decide on the connectivity that defines a Delaunay triangulation.

Consider the four nodes, 1,2,3 and 4 in Figure 2.8. If edges 1-2, 2-3, 3-4 and 4-1 are defined, there are two options to triangulate the resulting quadrilateral. Either edge 2-4 can be formed to give the two triangles 1-2-4 and 2-3-4, or edge 1-3 can be formed which results in the triangles 1-3-4 and 1-2-3. Which triangulation is Delaunay satisfying? The test is to apply the in-circle criterion. If a triangulation is Delaunay satisfying it conforms to the in-circle criterion and hence, a circle can be constructed through the three nodes which form a triangle and this circle must not contain any other node in the triangulation. It is apparent from Figure 2.8, that the two circles through the nodes of the triangles 1-2-4 and 2-3-4 do not enclose other nodes. However, the circle through the nodes of the triangle 1-3-4 contains node 2. Hence, the triangulation 1-3-4 and 1-2-3 does not satisfy the in-circle criterion and hence is not a Delaunay triangulation. On the other hand, triangles 1-2-4 and 2-3-4 do satisfy the in-circle criterion and hence form the Delaunay triangulation of the four nodes 1,2,3,4.

The Min-Max Property

The Delaunay triangulation maximises the minimum of the six angles in any pair of two triangles which make up a convex quadrilateral. This is best illustrated with respect to Figure 2.9.

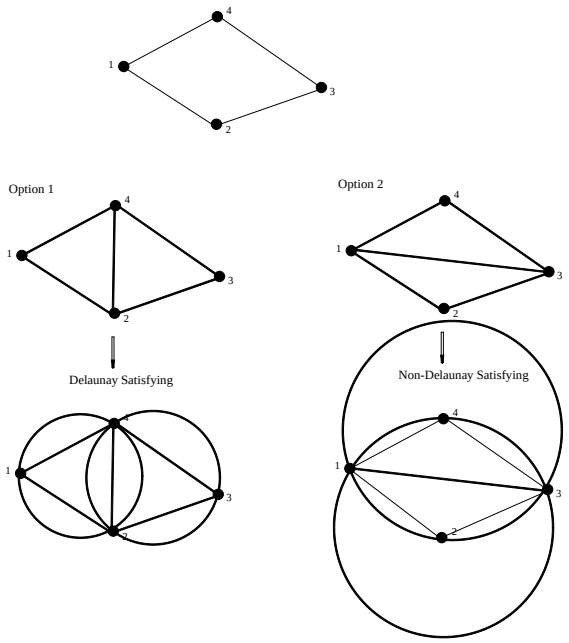

Figure 2.8: Delaunay criterion for four nodes

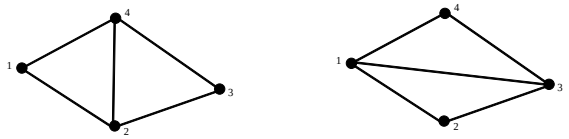

Figure 2.9: The connections show that Delaunay minimises the maximum angle

It follows from the in-circle criterion that the Delaunay triangulation is that formed from connecting nodes 1,2,4 and 2,3,4. It is noted that this triangulation produces the maximum minimum angle of the six angles in the two possible options.

Uniqueness of the Delaunay Triangulation

The Delaunay triangulation of an arbitrary set of nodes $P_i, i = 1, N$ is unique apart from degeneracies where nodes lie on the circumference of a circle.

Figure 2.10 shows four nodes and two Delaunay triangles. In the first two cases, the Voronoi vertices V_1 and V_2 associated with the two triangles are distinct. However, as the nodes P_1 and P_2 are moved closer together, the Voronoi edge connecting V_1 and V_2 is seen to decrease in length. Finally, the four nodes fall on the circumference of the same circle, the Voronoi vertices V_1 and V_2 become coincident and located, of course, at the centre of the two coincident circles which pass through the four nodes. At this degenerate stage, the strict application of the in-circle criterion allows for a valid triangulation in the form of the triangle P_1-P_2-P_3 and P_2-P_4-P_3 or an equally valid triangulation P_1-P_4-P_3 and P_1-P_2-P_4.

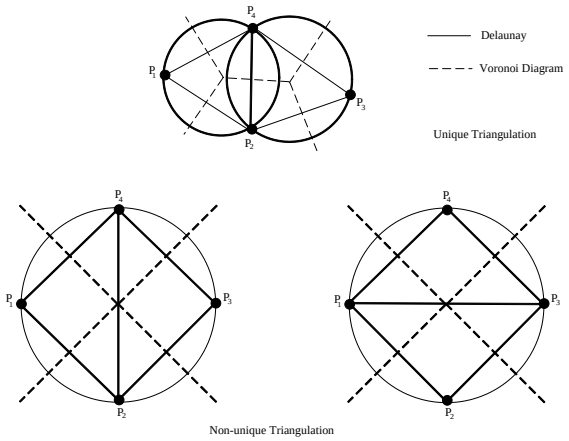

Figure 2.10: Uniqueness of the Delaunay triangulation

The triangulation, therefore, in this case is non-unique.

2.6.3 Construction of the Delaunay Triangulation

The details of the Delaunay triangulation and its dual the Voronoi diagram have been described. There are several reported methods to construct the triangulation. Given that it is proposed to use the Delaunay approach to build unstructured meshes for which millions of points may be required it is important that any method adopted should be fast and computationally efficient [11, 50, 100].

The Bowyer Algorithm

One of the most popular algorithms used to construct the Delaunay triangulation for mesh generation is that based upon the work of Bowyer. The original paper appeared in the Computer Journal in 1981 and was reported in the same issue as the paper by Watson which also addressed the issue of the construction of the Delaunay triangulation. The approach discussed by Bowyer is favoured over that of Watson because it applies equally in two and three dimensions.

The algorithm of Bowyer, which is based on the in-circle criterion, is a sequential process; each point is introduced into an existing Delaunay satisfying structure, which is locally modified to admit the new node and reconstructed into a form which is again Delaunay satisfying.

The basic outline of the algorithm, which is applicable in 2 dimensions is;-

1. Define the convex hull within which all points will lie. It is appropriate to specify 4 points, $P_i, i = 1, 4$ together with the associated Voronoi diagram structure.

2. Introduce a new point anywhere within the convex hull $P_i, i = 1, 4$.

3. Determine all vertices of the Voronoi diagram to be deleted. A point that lies within a circle, centred at a vertex of the Voronoi diagram and which

passes through its three forming points, results in the deletion of that vertex. This follows from the 'in-circle' definition of the Voronoi construction.

4. Find the forming points of all the deleted Voronoi vertices. These are the contiguous points to the new point.

5. Determine the Voronoi vertices which have not themselves been deleted which are neighbours to the deleted vertices. This data provides the necessary information to enable valid combinations of the contiguous points to be constructed.

6. Determine the forming points of the new Voronoi vertices. The forming points of new Voronoi vertices must include the new point together with the two points which are contiguous to the new point and form an edge of a neighbouring triangle (these are the possible combinations obtained from Step 5).

7. Determine the neighbouring Voronoi vertices to the new Voronoi vertices. Following Step (6), the forming points of all new vertices have been computed. For each new vertex, perform a search through the forming points of the neighbouring vertices as found in Step (5) to identify common pairs of forming points. When a common combination occurs, then the vertex is a neighbour of the Voronoi diagram.

8. Reorder the Voronoi diagram data structure, overwriting the entries of the deleted vertices.

9. Repeat steps (2-8) for the next point.

10. Delete unwanted triangles.

This algorithm provides a triangulation of the convex hull.

2.6.4 Issues of Implementation

The implementation of the algorithm is straightforward. The data structures used to implement the algorithm involves an array, *ElementConnectivity*, which contains the nodes of each triangle (the so-called forming points) and a data structure *AdjacentElements*, which, for each triangle, points to the neighbour triangle. In fact, since there is a Voronoi vertex associated with each triangle, this is equivalent to pointing to the adjacent Voronoi vertices. The circumcentre, *VoronoiCentre*, and radius, *Radius*, of each circle which passes through the three nodes of each triangle are also required. In addition, obviously the co-ordinates of the points, *PointCoordinates* must be stored. Hence, the memory requirements, therefore, are 12 times the maximum number of triangles, and three times the maximum number of points.

Table 2.1 shows this data structure for the geometrical construction shown in Figure 2.11. The zeros in the table indicate that the neighbour vertices are not defined, i.e. they are outside the convex hull.

Table 2.1: Data Structure for Delaunay construction

$Vertex$	$FormingNodes$	$NeighbourVertices$				
1	1	2	3	2	0	0
2	2	3	4	1	3	0
3	3	4	9	2	4	0
4	4	7	9	3	5	6
5	7	8	9	4	7	0
6	4	7	6	4	8	0
7	5	7	8	5	8	0
8	5	6	7	6	7	0

In relation to each steps in the algorithm, the following comments can be made:

Step 1 The data structure required to define the convex hull, within which all nodes are contained, is set up in a subroutine. The co-ordinates of the four nodes which define the region are computed when all the boundary nodes are available. The nodes of the convex hull points are then scaled in an appropriate way. Figure 2.11 shows the initial set-up and Table 2.2 shows the relevant data structure.

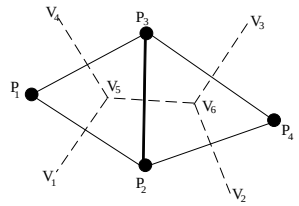

Figure 2.11: Initial set-up for the Delaunay construction in two dimensions

Table 2.2: Initial data structure

$Vertex$	$FormingNodes$	$NeighbourVertices$				
1	0	1	3	0	0	5
2	0	3	4	0	0	6
3	0	2	4	0	0	6
4	0	1	2	0	0	5
5	1	2	3	1	4	6
6	4	2	3	5	2	3

Step 2 The boundary nodes are assumed known and are input in a sequentially into the Delaunay algorithm.

Step 3 This involves two basic operations. Firstly, it is necessary to determine whether a triangle should be deleted if the new node is contained within the circle associated with the triangle. Secondly, there is a search to determine all the

triangles which should be deleted. The search for Voronoi vertices to be deleted is potentially a computationally expensive procedure and attention must be given to a fast and efficient search routine

The in-circle criterion implies that a triangle of an existing Delaunay triangulation will be deleted if the node is contained within the circumcircle passing through the three nodes of the triangle. Hence, a new node with co-ordinates *xnew*, will lie within a circle of radius r, and circumcircle position xc if

$$|X_c - X_{new}| < r \tag{2.8}$$

where —— is the Euclidean distance. This would appear to be a simple test to perform. However, numerical arithmetic on a computer is not exact and for some cases,

$$|X_c - X_{new}| < \varepsilon \tag{2.9}$$

where ε is the round-off of the computer. In this case it is not possible to determine if a node is inside or outside a circle, Figure 2.12.

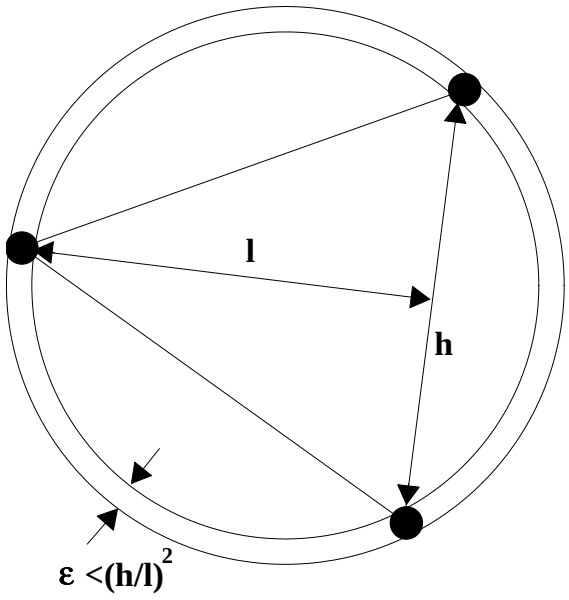

Figure 2.12: Voronoi tolerance band

However, if a new node is introduced which lies within the tolerance band then it will be impossible to ensure that it is correctly checked. Typically, 64-bit arithmetic must be used.

Figure 2.12 Simplistically, a worst case search for all triangles to be deleted would involve the application of this test for every existing circle. However, the search can be made more efficient by utilising the tree data structure. On finding a triangle, say τ_0, whose circumcircle encloses the new point, a tree search is implemented to find all others. Condition 8.1 is applied to the neighbours of τ_0 i.e.

$$|VoronoiCentre(AdjacentElements(\tau_0, i)) - X_{new}| < Radius(AdjacentElements(\tau_0, i)), i = 1, 2 \tag{2.10}$$

The search is again continued into the neighbours only if a Voronoi region $AdjacentElements(\tau_0, i), i = 1, 3$ satisfies equation (7.1). The search terminates when no neighbour regions to the affected regions satisfy test (7.1). It is clear that the use of the tree data structure will, in most cases, be more cost effective than the search through all vertices. The search time, therefore, is dependent upon the time required to find the first circle which contains the new node. When triangulating the initial set of boundary nodes it proves effective to begin the search in the last triangle to be created and then to search backwards through the list. When triangulating the interior field nodes, it will be shown that the automatic node insertion routine provides a convenient way of specifying the triangle in which to begin the search.

As an example consider Figure 2.13. The new point is inserted into the triangulation and falls within the triangle, O. The in-circle test is then performed on the neighbour triangles A, B and C. The node lies within the circle for triangle A and hence, in turn, its neighbours, triangles D, E, are tested. (Since the neighbour triangle O has already been tested it is not necessary to perform this test again.) The new point does not fall inside the circle associated with triangle B and therefore it is not necessary to consider the neighbour triangles. However, the new point does fall inside the circle for triangle C and hence its neighbour triangles F and G are tested. On testing the neighbours of the neighbours, i.e. triangles D, E, F and G, it is found that none pass the incircle test. Hence, the triangles to be deleted are triangles O, A and C.

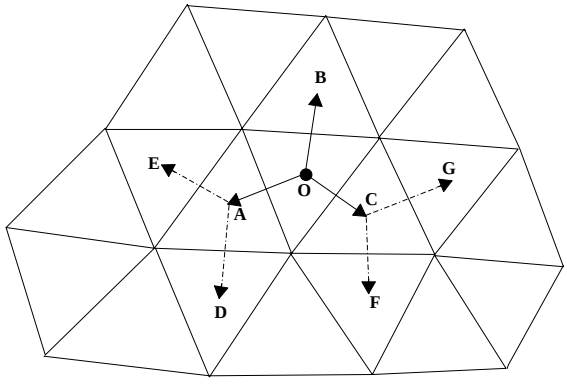

Figure 2.13: Local search to finds triangles to be deleted

Step 4 The forming nodes of the deleted Voronoi vertices are obtained by forming the unique list of nodes from the deleted vertices i.e.

$$ElementConnectivity(DeletedtTriangle(k), i), k = 1, K, i = 1, 3 \tag{2.11}$$

where K is the number of triangles identified from Step 3. These nodes, $NODES(j), j = 1, M$, surround the new node X_{new} and form a convex region.

Step 5 When performing the tree search to find all Voronoi vertices to be deleted it was noted that neighbours to deleted vertices were also visited. It is a simple task, during this search to flag all vertices which were visited but found subsequently on test (8.3.1) not to be deleted. The list of such vertices, with nodes, $NNODES(j), j = 1, L$, are the neighbour vertices to the deleted polygon and, therefore, they must be neighbours to the new triangles created.

Step 6 The new triangles are formed by the new node and permutations of 2 nodes of $NODES(j), j = 1, M$. The correct permutations are those which occur in doubles in the neighbour vertices determined in Step 5.

Step 7 The neighbour data is obtained by finding common doubles in the nodes, $NODES(j), j = 1, M$, and $NNODES(j), j = 1, L$.

Step 8 The newly created triangles are put into positions in the data structure of the old deleted triangles, and once these have been filled are added to the end of the list of triangles. Similarly, with the entries into the tree data structure.

Step 9 Having completed Steps 1-8 the new node is now included in the triangulation which is fully Delaunay satisfying and the full data structure of the new construction is complete. Hence, the next new point can be taken and the algorithm repeated from Step (ii).

2.6.5 Node Creation

As stated in Section 8.1, the Delaunay algorithm is a procedure to form a triangulation from a given set of nodes. The technique does not provide a means of node co-ordinate generation. Nodes for connection by the Delaunay algorithm can be derived in many ways.

Automatic Point Creation driven by the Boundary Point Distribution

The computational domain is defined in discrete form by the boundary points. It will be assumed that this point distribution reflects appropriate geometrical features, such as variation in curvature and gradient. When the boundary points are inserted into the Delaunay algorithm a triangulation of the domain is created. A class of point creation methods can be defined which are designed to refine this triangulation until some quality criterion is satisfied. Initially, consider the generation of points to produce a regular triangulation, i.e. an isotropic triangulation.

Consider, in two dimensions, boundary line segments on which points have been distributed which enclose a domain. It is required to distribute nodes within the region so as to construct a smooth distribution of nodes. For each point on the boundary, a typical length scale for the point can be computed as the average of the two lengths of the connected edges. The basic idea is to refine the Delaunay triangulation of the boundary points until elements are formed with length scales consistent with those computed for all boundary nodes. How the triangles are refined distinguishes the many different approaches that have been investigated. Commonly used procedures include

1. Node placement at circumcentres of triangles which violate the spacing criterion

2. Node placement at centroids of traingles which violate the spacing criterion.

3. Node placement along edges of elements which violate the spacing criterion.

Once a point has been inserted, it too must have an associated length scale in order for the interative refinement to continue. A newly inserted node takes a length scale from interpolation of the length scales from the nodes which formed the triangle from which it was created. In this way, a smooth transition between boundaries of interior points can be ensured. This process of point insertion continues until no point can be added because the given quality criterion is satisfied.

In further detail, an algorithm is presented which describes the basic principle and uses point placement at centroid positions.

1. Compute the point distribution function, dp_i,for each boundary point $r_i = (x, y)$, i.e. for point o

$$dp_i = \sum_{i=1}^{M} |r_i - r_o|/M \tag{2.12}$$

   where —— is the Euclidean distance and it is assumed that point o is surrounded by M points.

2. Generate the Delaunay triangulation of the boundary points.

3. Initialize the number of interior field points created, $N = 0$.

4. For all triangles within the domain,

   (a) Define a prospective point, Q, to be at the centroid of the triangle.

   (b) Derive the point distribution, dp, for the point Q, by interpolating the point distribution function dp_m , $m = 1, 2, 3$. from the nodes of the triangle, which contains Q.

   (c) Compute the distances d_m , $m = 1, 2, 3$ from the prospective point, Q, to each of the points of the triangle.

      i. If $d_m < \alpha * dp_m$ for any $m = 1, 2, 3$ then reject the point :-Return to the beginning of step (iv)

      ii. If $d_m > \alpha * dp_m$for any $m = 1, 2, 3$ then compute the distance s_j , $j = 1, N$, from the prospective point Q, to other points to be inserted, P_j, $j = 1, N$.

      iii. If $s_j < \beta * dp_m$ then reject the point :- Return to the beginning of step (iv).

      iv. If $s_j > \beta * dp_m$ then accept the point Q for insertion by the Delaunay triangulation algorithm. Include Q in the list P_j , $j = 1, N$.

   (d) Assign the interpolated value of the point distribution function, dp, to the new node, PN .

   (e) Next triangle.

5. If $N = 0$ go to step (vii).

6. Perform Delaunay triangulation of the derived points, P_j , $j = 1, N$. Go to step (iii)

7. Smooth the mesh.

The coefficient α controls the mesh point density, whilst β has an influence on the regularity of the triangulation. It should be noted that this point creation algorithm can be implemented very efficiently within the Delaunay triangulation procedure. In particular, if a point is accepted for insertion it must be contained with a triangle in the Delaunay construction. The number of the triangle is flagged with the new node. This triangle is the starting triangle for the search to find all triangles to be deleted by the in-circle test. However, after the insertion of one point the numbering of the triangles can change and if the triangle formed from the inserted points overlap then the triangle numbers which have been flagged for each new point can then be incorrect. However, the exclusion zone, controlled by the parameter β ensures that on the insertion of each point the resulting triangulation does not overlap and hence the original numbered triangles associated with each new point are valid. Hence, in this way, β improves the regularity of the triangles and also ensures that no search is required to find a circle which includes the new point for insertion.

2.6.6 Boundary Recovery in Two Dimensions

As was briefly discussed in the introductory comments, one of the basic requirements of the mesh is for it to be boundary conforming. In the Delaunay triangulation, boundary conformity has to be checked and if necessary enforced by the use of special techniques. Given a set of points which describe a geometry in two dimensions and the Delaunay connections between these points there is no guarantee that the resulting triangulation will contain edges or faces which conform to the boundary surface. In fact, for complicated shapes, the boundary edges and faces will almost certainly not be recovered. The techniques devised to correct the triangulation or force boundary integrity are many.

2.6.7 Extension of the Delaunay Procedure to Three Dimensions

This criterion forms the basis for the most popular algorithm for the construction of the tessellation which was proposed by Bowyer and Watson.

The basic outline of the algorithm [101, 102], which is applicable in 3 dimensions is;-

1. Define the convex hull within which all points will lie. It is appropriate to specify eight points together with the associated Voronoi diagram structure.

2. Introduce a new point anywhere within the convex hull.

3. Determine all vertices of the Voronoi diagram to be deleted. A point which lies within the sphere, centred at a vertex of the Voronoi diagram and which passes through its four forming points, results in the deletion of that vertex. This follows from the 'in-circle' definition of the Voronoi construction.

4. Find the forming points of all the deleted Voronoi vertices. These are the contiguous points to the new point.

5. Determine the Voronoi vertices which have not themselves been deleted which are neighbours to the deleted vertices. These data provide the necessary information to enable valid combinations of the contiguous points to be constructed.

6. Determine the forming points of the new Voronoi vertices. The forming points of new vertices must include the new point together with the three points which are contiguous to the new point and form a face of a neighbouring tetrahedra (these are the possible combinations obtained from Step v).

7. Determine the neighbouring Voronoi vertices to the new Voronoi vertices. Following Step vi, the forming points of all new vertices have been computed. For each new vertex, perform a search through the forming points of the neighbouring vertices as found in Step v to identify common pairs of forming points. When a common combination occurs, then the three associated vertices are neighbours of the Voronoi diagram.

8. Reorder the Voronoi diagram data structure, overwriting the entries of the deleted vertices.

9. Repeat steps (ii-viii) for the next point.

2.6.8 Edge and triangle recovery in arbitrary constructions of tetrahedra

The general problem faced in three dimensions is to recover a given set of edges and triangles within an assembly of tetrahedra, i.e. to recover a set of edges $E_i, i = 1, I$ formed by the node connectivities $n_k^i, k = 1, 2$ and to recover a set of triangles $T_j, j = 1, J$ formed by the node connectivities $n_k^j, k = 1, 3$. We will assume that the nodes $n_k^i, k = 1, 2$ and $n_k^j, k = 1, 3$ are given and are contained within the triangulation $\tau_l^3, l = 1, L$

To definition 1, we include the definition of a triangle: *Definition 2.* A triangle consists of the region bounded by three edges, formed by connecting three nodes A, B and C in cyclic order.

The necessary and sufficient condition for an edge with nodes A and B to be present in the triangulation $\tau_l^3, l = 1, L$ is:

1. The nodes A and B are contained in an element with node connectivity , $(n_1, n_2, n_3, n_4)^l$, within the tetrahedral construction $\tau_l^3, l = 1, L$. This condition implies that A and B exist in the tetrahedral construction.

   The necessary and sufficient condition for a triangle with nodes A, B, C to be present in the triangulation $\tau_l^3, l = 1, L$ is:

2. The cyclic combinations of A, B and C, i.e. $A - B$, $B - C$, $C - A$ occur in an element with node connectivity $(n_1, n_2, n_3, n_4)^l$, within the tetrahedral

construction $\tau_l^3, l = 1, L$. Again condition (2) implies that A, B and C exist in the tetrahedral construction.

Edge recovery

The procedure to recover a missing edge involves two steps. First, it is necessary to identify the triangles, edges and points of tetrahedra which the edge intersects. Second, local transformations involving tetrahedra are performed to recover the edge.

Consider an edge joining two points A and B which is not contained in the tetrahedral construction. The line does not exist because triangles, edges or points of tetrahedra intersect the edge $A - B$. The line segment from point A can intersect the tetrahedron containing A, in the direction of $A - B$, through a triangular face, edge or node. In turn, the next line segment in the adjacent tetrahedron can intersect through a triangle face, edge or node and so on through to the tetrahedron which contains node B. Hence, it is possible for the edge $A-B$ to pass through tetrahedra with a combination of intersections, which include:

node to node edge to node face to node

node to edge edge to edge face to edge

node to face edge to face face to face

The edge $A-B$ can intersect many tetrahedra but the intersection types must fall into one of the categories described above.

Having established the intersection types for each intersected tetrahedra for the line segment $A - B$, it is possible to then perform direct transformations to recover the edge. A transformation denoted by $(1 - 3)_{n,f}$, for example, takes one tetrahedron to three tetrahedra for a node-to-triangular face intersection of a line with a tetrahedron. Transformation of a node-to-edge intersection, $(1 - 2)_{n,e}$ requires the two tetrahedra with the triangular face common to the edge segment each to be subdivided into two, with the introduction of a new node, corresponding to the intersection point of the line segment $A - B$ with the intersected edge. Transformation of an edge-to-edge intersection, $(1 - 3)_{e,e}$ requires each of the two tetrahedra with the face common to the edge segment to be subdivided into three. Two new points are introduced, again corresponding to the intersection points of the line segment $A - B$ with the two intersected edges. Similar procedures apply to the transformation $(1 - 3)_{n,f}$, $(1 - 5)_{f,f}$ and $(1-4)_{e,f}$ which involve the creation of three, five and four tetrahedra, respectively, to recover the line segment with intersections of node-to-face, face-to-face and edge-to-face.

It is noted that if the combination of intersections involves a node-to-face and a face-to-node, the edge $A - B$ can be recovered directly by transforming two tetrahedra into three tetrahedra.

Boundary face recovery

The recovery of a specified triangle $A-B-C$ is achieved in a two phase process. First, edges of the triangle, namely, $A - B$, $B - C$, $C - A$ are recovered using the techniques described in the previous section. Once these edges are present, a similar procedure follows to recover the triangle. If the edges $A - B$, $B - C$ and

$C - A$ exist but the triangle $A - B - C$ does not exist, then all tetrahedra which possess at least one edge which intersects the triangle $A - B - C$ are determined.

These tetrahedra are then modified accordingly to recover the missing triangles whose union forms the required triangle $A - B - C$.

One, two, three or four edges of a tetrahedron can intersect a triangle. Hence, for each missing triangle, all tetrahedra which have an edge or edges which intersect the triangle are determined and each of the tetrahedra is then classified accordingly. For each of the four types of intersections, transformations can be used to recover the triangle.

For a tetrahedron with one intersecting edge, a node is inserted at the intersection of the edge and the triangle. New connections are then formed. Similar procedures are required for the other transformations. It is noted that if one, and only one, edge intersects a triangle and that edge is common to three tetrahedra then the triangle can be recovered directly by deleting the edge and thus forming the required triangle. This results in three tetrahedra collapsing to two tetrahedra with the recovery of the triangle.

2.7 Hybrid Unstructured Mesh Generation

The standard Delaunay/advancing–front approach described generates valid simplex meshes for complex three–dimensional geometries in a matter of hours. Compared to structured hexahedral meshes of similar node spacing however, the connectivity of the unstructured mesh is significantly higher. It can be shown that the computer resources required to obtain a solution on a given mesh is approximately proportional to the number of edges in the mesh. For a tetrahedral mesh, the number of edges is about 6.5 times the number of nodes, while for large structured hexahedral meshes, there are only about 3 edges per node. This means that more than twice the time for each right–hand side evaluation is required for the tetrahedral mesh, and, in addition, more memory is needed to store the data for the extra edges.

This problem is usually more critical for viscous flows, where the mesh sizes are usually at least one order larger than for inviscid calculations on the same geometry. For such meshes, it is possible to take advantage of the way the unstructured meshes are created close to viscous boundaries to reduce the connectivities of the mesh. The advancing layer method creates meshes that are of structured appearance in the normal direction of the viscous wall. This regularity can be exploited in the generation of hybrid meshes. In the current work, edges are removed in such layers by merging elements. This implementation was performed by the authors supervisors and is also described in [36]. Other implementations of hybrid meshes are described in [34, 71, 72, 99]. Since most of the elements are situated in the viscous layers for high–Reynolds number flows, a reduction of edges in these layers will yield significant global edge reductions. Typically the number of edges of a given tetrahedral mesh can be reduced by $30 - 45\%$.

In addition to the substantial reduction in memory and CPU usage, the diagonals of simplex meshes appearing in viscous layers can have adverse effects on the solution accuracy in these regions. It is, therefore, also desirable from an

accuracy point of view to eliminate the diagonal edges in the wall layers. The improvements in solution accuracy that can result from the introduction of hybrid meshes is shown in Figure 4.10 and has also been reported by other authors [56].

The hybrid mesh generation procedure used in three dimensions may be described as follows:

1. Quadrilateral elements are generated on the wall surfaces. This is accomplished by an indirect method of combining triangles, coupled with a splitting scheme to guarantee meshes consisting of quadrilateral elements only. The major steps of this process are:

   (a) The advancing front technique is employed to generate a triangular mesh of double the desired element size.

   (b) Triangles are combined to form quadrilateral elements, using an enhancement to Lo's strategy proposed by Lee [49]. An initial front is defined to consist of the edges of triangles at the boundary of the surface domain. Triangles are systematically combined at the front, advancing towards the interior of the area. The front defines the boundary between the region of quadrilaterals and the region of triangles yet to be combined. The sides in the front are ordered in an ascending manner according to the number of triangles which have not been combined and share a node with the side. With this technique it is possible to guarantee a final mesh of quadrilaterals only, provided the initial number of edges on the boundary is even.

   (c) Mesh cosmetic techniques are then employed to improve the quality of the quadrilateral elements. This ensures that no adjacent elements have two common faces, that no quadrilateral element is surrounded by four elements and that no side has both nodes connected to three quadrilaterals.

   (d) Element splitting is employed to guarantee a mesh of quadrilaterals only. The process divides triangular elements into three quadrilaterals, quadrilateral elements into four quadrilaterals and a pentagon formed from a quadrilateral adjacent to a triangle into five quadrilaterals.

   (e) A Laplacian smoothing algorithm is employed to improve the mesh quality.

2. The advancing layer method [33] is employed to generate stretched elements adjacent to those boundary surface components which represent solid walls. The height and number of layers is specified by the user in such a way that the expected boundary layer profile is adequately resolved. The implementation starts from a triangular mesh on the surface of the wall, which means that each quadrilateral element has to be treated as two triangles in the generation process. The layers are constructed by generating points along prescribed lines and connecting the generated points, by using advancing front mesh generation concepts, to form tetrahedral elements. For the first layer, the prescribed lines are normal to the surface, while for succeeding layers, smoothing of the line directions is employed. Surface intersections

require special treatment [33]. Point generation ceases before the prescribed number of layers is reached if an intersection appears or if the local mesh size is close to that specified in the user–specified mesh distribution function.

3. The remainder of the domain is meshed using a standard isotropic Delaunay procedure as described in section **??**.

4. Tetrahedra are merged to form prisms and pyramids. Starting at the boundary, the concept employed while generating the advancing layers is used to determine the point grown from each boundary point. For each layer, each boundary triangle is considered in turn and a prism is generated if all three vertices of the triangle have generated new points, a pyramid is generated if two of the vertices have successfully generated new points and a tetrahedron will be generated if there is only one vertex generating a new point.

5. Hexahedral elements are generated by using the fact that pairs of triangles on the wall are the result of splitting each of the original quadrilateral elements generated on the surface. The two prisms constructed on these two triangles will be combined to form a hexahedron. This combination can only be performed if the two tetrahedra in the isotropic mesh containing the two triangles grown from the original quadrilateral surface elements can be merged into a pyramid. When this condition is not satisfied, it is possible to overcome the problem by inserting a point at the centroid of the hexahedron of the final layer and then splitting the hexahedra into five pyramids and two tetrahedra.

For two–dimensional meshes, the quadrilateral elements are generated close to viscous walls by merging two triangles. An example of a hybrid mesh in two–dimensions is given in Figure 2.14

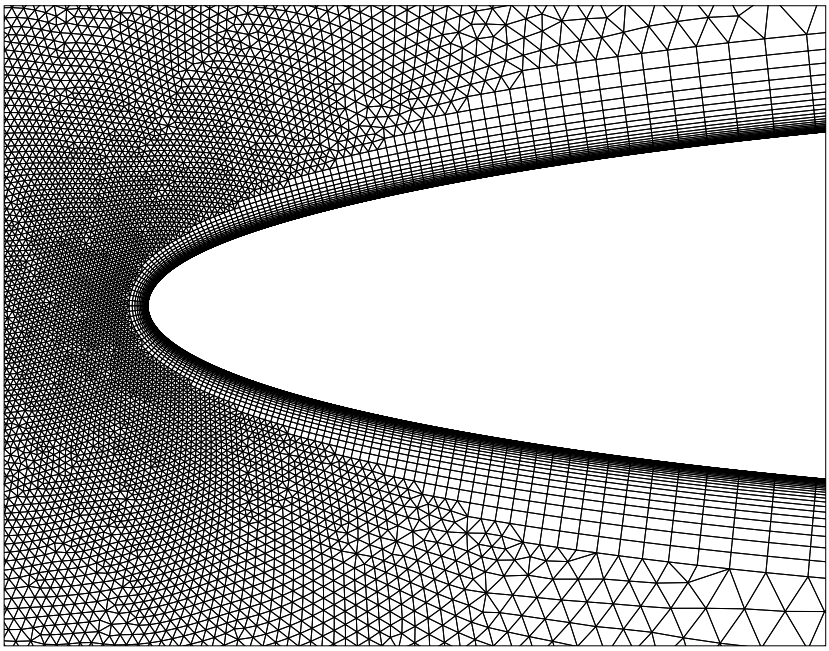

Figure 2.14: Detail of a hybrid two–dimensional mesh.

Chapter 3

Problem Formulation

3.1 Introduction

The fluid flow problems discussed in this project are governed by the compressible Euler and Navier–Stokes equations in their stationary and time–dependent forms under the assumption that the fluid is a calorically perfect gas. In this chapter, a summary of these formulations is presented. In addition, the time–averaging procedure used for turbulent flows is explained and the turbulence model used in this work is summarized. A short discussion of the boundary conditions relevant to these formulations is also presented.

3.2 Governing Equations

The time–dependent, compressible Navier–Stokes equations in integral form on a three–dimensional Cartesian domain $\Omega \subset \mathbb{R}^3$, with surface $\partial\Omega$, can be expressed as

$$\int_\Omega \frac{\partial U_i}{\partial t} \mathrm{d}\boldsymbol{x} + \int_{\partial\Omega} F_{ij} n_j \mathrm{d}\boldsymbol{x} = \int_{\partial\Omega} G_{ij} n_j \mathrm{d}\boldsymbol{x}, \tag{3.1}$$

where n_j is the outward unit normal vector to $\partial\Omega$ and the unknown vector of the conservative variables is given by

$$\boldsymbol{U} = \begin{bmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ \rho u_3 \\ \rho \epsilon \end{bmatrix}. \tag{3.2}$$

Here ρ denotes the fluid density, u_i the i'th component of the velocity vector and ϵ the specific total energy. The inviscid and viscous flux tensors are defined by

$$\boldsymbol{F}_j = \begin{bmatrix} \rho u_j \\ \rho u_1 u_j + p\delta_{1j} \\ \rho u_2 u_j + p\delta_{2j} \\ \rho u_3 u_j + p\delta_{3j} \\ u_j \left(\rho\epsilon + p\right) \end{bmatrix} \tag{3.3}$$

and

$$\boldsymbol{G}_j = \begin{bmatrix} 0 \\ \tau_{1j} \\ \tau_{2j} \\ \tau_{3j} \\ u_k\tau_{kj} - q_j \end{bmatrix} \tag{3.4}$$

respectively. In the above,

$$\tau_{ij} = -\frac{2}{3}\mu\frac{\partial u_k}{\partial x_k}\delta_{ij} + \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{3.5}$$

is the deviatoric stress tensor, μ is the dynamic viscosity coefficient, and

$$q_j = -k\frac{\partial T}{\partial x_j} \tag{3.6}$$

is the heat flux where k is the thermal conductivity and T is the absolute temperature. It is assumed that the viscosity varies with temperature according to Sutherland's law,

$$\frac{\mu}{\mu_0} = \left(\frac{T}{T_0}\right)^{3/2}\frac{T_0 + 110}{T + 110}. \tag{3.7}$$

where the temperature is expressed in degrees Kelvin and (μ_0, T_0) is a reference state . It is assumed that the Prandtl number

$$Pr = \frac{c_p\mu}{k} \tag{3.8}$$

is constant and equal to 0.72. In this expression, c_p is the specific heat at constant pressure. The system is closed by assuming the gas to be calorically perfect, thus setting

$$p = \rho RT \tag{3.9}$$

and

$$\epsilon = c_v T + \frac{1}{2}u_k u_k \tag{3.10}$$

where R is the real gas constant and $c_v = c_p - R$ is the specific heat at constant volume. Throughout this work, the ratio of the specific heats,

$$\gamma = \frac{c_p}{c_v} \tag{3.11}$$

is set to $\gamma = 1.4$ which is the value for air at standard conditions.

The above formulation is only valid for continuous media, and is therefore not applicable to problems where the gas density is low, such as initial stages of reentry. It is also assumed that there is no radiation of heat into the domain considered and that no chemical reactions take place in the gas. Body forces have been neglected. In particular, gravity is not taken into account, as its influence is regarded as negligible for the problems treated in this work.

3.2.1 Dimensionless Parameters

In Appendix **??**, the governing equations are nondimensionalized. It is shown that equation (3.1) is invariant with respect to the Prandtl, Reynolds, Mach, and Strouhal numbers given by

$$Pr_\infty = \frac{c_p \mu_\infty}{k_\infty} \tag{3.12}$$

$$Re_\infty = \frac{\rho_\infty u_\infty l}{\mu_\infty} \tag{3.13}$$

$$M_\infty = \frac{u_\infty}{c_\infty} \tag{3.14}$$

$$St_\infty = \frac{u_\infty t_c}{l} \tag{3.15}$$

respectively, where u is the velocity norm, l is some specified reference length scale, c is the speed of sound and t_c is some characteristic time scale. The subscript ∞ denotes freestream values, but the equations can also be scaled by another constant state. For a given geometry with corresponding boundary conditions, the flow is fully described by these four parameters, allowing the unknowns to be scaled to reduce the numerical errors generated in the computations.

It is often interesting to calculate the total lift, drag and moment of a given configuration. These values are usually specified in terms of the dimensionless coefficients

$$C_l = \frac{1}{q_\infty S_\Gamma} \int_\Gamma \boldsymbol{\sigma}_j^w n_j^\infty \mathrm{d}\boldsymbol{x} \tag{3.16}$$

$$C_d = \frac{1}{q_\infty S_\Gamma} \int_\Gamma \boldsymbol{\sigma}_j^w t_j^\infty \mathrm{d}\boldsymbol{x} \tag{3.17}$$

$$C_{m,i} = \frac{1}{q_\infty S_\Gamma l} \int_\Gamma \theta_{ijk} \sigma_j^w r_k \mathrm{d}\boldsymbol{x} \tag{3.18}$$

for the lift, drag and moment respectively, where Γ is the integration surface. The freestream dynamic pressure is given by

$$q_\infty = \frac{1}{2} \rho_\infty u_\infty^2 \tag{3.19}$$

and S_Γ is a reference area, taken as the wing planform for aircraft configurations. The freestream flow tangent is defined by

$$t_j^\infty = \frac{u_j^\infty}{u_\infty} \tag{3.20}$$

and n_j^∞ is a specified freestream flow normal, which is normal to t_j^∞. Equation (3.16) also introduces the wall stress vector

$$\sigma_i^w = \left(-p\delta_{ij} + \tau_{ij}\right) n_j^w \tag{3.21}$$

where n_j^w is the wall normal. For the moment coefficient, a length scale l has to be defined. For wings, this is usually taken to be a reference chord length. In addition, a reference line or point from which the momentum is taken must be

defined and the position vector r_j is defined as the shortest distance vector from this geometrical reference.

In addition to integrated entities, it is sometimes useful to introduce certain pointwise dimensionless coefficients. Two commonly used parameters are the pressure coefficient and skin friction coefficient defined by

$$C_p = \frac{p - p_\infty}{q_\infty} \tag{3.22}$$

and

$$c_f = \frac{\psi}{q_\infty} \left|\left|\tau_i^s\right|\right|, \tag{3.23}$$

respectively. The shear stress vector on the surface is given by

$$\tau_i^s = \tau_{ij} n_j^w - \tau_{jk} n_j^w n_k^w n_i^w \tag{3.24}$$

and ψ is a sign function, defined by

$$\psi = \begin{cases} t_i^\infty \tau_i^s / \left| t_j^\infty \tau_j^s \right|, & t_j^\infty \tau_j^s \neq 0 \\ 0, & t_j^\infty \tau_j^s = 0. \end{cases} \tag{3.25}$$

3.2.2 Reduced Formulations

Often it is convenient, or necessary, to simplify the original set of equations to forms more easily solved. If the fluid is assumed inviscid, the viscous terms in equation (3.1) are removed, leading to the reduced equation

$$\int_\Omega \frac{\partial U_i}{\partial t} \mathrm{d}\boldsymbol{x} + \int_{\partial\Omega} F_{ij} n_j \mathrm{d}\boldsymbol{x} = 0. \tag{3.26}$$

This system is referred to as the Euler equations and is known to give good pressure distributions for many problems where the boundary layers are thin. If time–accuracy is not needed, the steady–state version of equation (3.1) may be solved in the form

$$\int_{\partial\Omega} F_{ij} n_j \mathrm{d}\boldsymbol{x} = \int_{\partial\Omega} G_{ij} n_j \mathrm{d}\boldsymbol{x}, \tag{3.27}$$

and

$$\int_{\partial\Omega} F_{ij} n_j \mathrm{d}\boldsymbol{x} = 0 \tag{3.28}$$

is solved for steady–state inviscid flow.

3.3 Turbulence Modeling

The governing equations described in the previous section can be considered exact under the assumptions stated. They are therefore capable of accurately reproducing physical flow patterns in a great range of cases where the physics are not

too severe and do not violate the assumptions made. In particular, the equations should be capable of predicting turbulence, at least the time–averaged solution of these highly chaotic and unsteady flows. Indeed, the governing equations have been solved directly for simple turbulent flows using *direct numerical simulation* (DNS) [44, 79]. But, as shown in [91], even though turbulence is a continuum phenomena for aerodynamic applications, it inherently includes chaotic flowfields at very small length scales, thus requiring an extremely fine numerical discretization to accurately capture the flow. According to Kolmogorov [46], there are characteristic length scales in turbulence down to a microscale defined by

$$\eta = \left(\frac{\mu^3}{\chi \rho^3} \right)^{1/4} \tag{3.29}$$

where χ is the dissipation rate of turbulence per unit mass. It can be shown [91] that the relation between η and the molecular mean free path of a gas, ξ, follows the relation

$$\frac{\xi}{\eta} \sim \frac{M}{Re_l^{1/4}} \tag{3.30}$$

where the Reynolds number, Re_l is defined with respect to the integral turbulence length scale which represents the length of the largest eddies in the flow. From the kinetic theory of gases, it is known that

$$\xi \sim \frac{\mu}{c\rho}. \tag{3.31}$$

By using these relations, and assuming a flow at standard sea level conditions around an aircraft flying at the speed of sound with $Re_l = 10^5$, one gets $\eta \sim 10^{-6} m$. For an aircraft with a turbulent wetted area of, say, $100 m^2$, and with an average turbulent boundary layer thickness of $0.01 m$, there would be a need for in the order of 10^{18} discrete points in the numerical calculation, excluding wakes. Clearly, this number of calculation points cannot be achieved on any existing computer memorywise, and a capability of solving time–accurately a system of 5×10^{18} unknowns over the period of time required to perform statistical averaging is not likely to exist in the near future.

Due to this complexity of DNS for realistic aerodynamic flows, researchers are forced to approximate the turbulent flow in some sense to reduce the computational effort needed. This modeling can be done at various levels, depending upon how much of the original Navier–Stokes equations are retained. One approach is to model the microscale turbulence using a semi–analytical or empirical approach, while still using the original system of equations to calculate the large–scale eddies. This *large eddy simulation* (LES) approach takes advantage of the fact that the mean microscale turbulence is reasonably homogenous and therefore easier to model than the large scale chaotic flow. In addition, the time scales of the turbulence on these length scales are very small and are therefore not usually interesting in the application when compared to the larger scale chaos which can induce flutter and other relevant time–dependent phenomena.

While LES has shown promising results for simple flows [23], the approach is still computationally too demanding for turbulent calculations on typical aerospace

applications. It has been noted in [103] that the CPU time required for LES calculations performed at present is typically five to ten percent of that needed for DNS. The dominant approach in industrial use today is therefore full turbulence modeling, where turbulence at all length scales is approximated by a given model. Within full turbulence modeling one usually separates between Reynolds–stress and turbulent viscosity models, the latter being the simplest and by far the most applied at present. For Reynolds–stress models, the correlation product $\overline{\rho u'' u''}$ is directly modeled while for the turbulent viscosity approach the Boussinesq assumption is introduced, treating the turbulence effects as the normal laminar diffusive terms in the Navier–Stokes equations. At the moment, it is unclear whether Reynolds–stress models yield more accurate predictions of turbulence compared to the implementationally and computationally simpler turbulent viscosity counterpart [80]. In this work, the turbulent viscosity approach is considered exclusively.

There are several turbulent viscosity models in use today. The different models are usually divided according to the number of differential equations used to model the turbulent viscosity. A very simple choice is an algebraic turbulence model, also termed a zero equation model, that does not require the solution of differential equations to produce the turbulence viscosity. These approaches are usually inspired by Prandtl's mixing–length theory [91] where the turbulent viscosity is related to momentum transfer in shear layer flows. The most commonly used methods of this class are the Cebeci–Smith [87] and Baldwin–Lomax [3] models. These models can give good results for flows involving small separation and simple geometries, but have clear limitations on their applicability for more complex problems.

A significant improvement in applicability can be achieved by introducing differential equations into the model. Two popular one–equation schemes are the Baldwin–Barth [2] and Spalart–Allmaras [88] models. They were constructed for use in aerodynamic applications and produce good results for a wide variety of problems within this category [4]. One drawback of the Spalart–Allmaras model is the need for the user to specify trigger regions where the laminar flow becomes turbulent. It is argued [88] that most turbulence models are incapable of finding such separation regions anyway, and that the trigger line approach can easily be connected to a separation detection model eliminating the need for user input. It has been stated [64] that the Baldwin–Barth model exhibits grid dependency and concluded [85] that the Spalart–Allmaras model is superior to this model. It, therefore, seems as though the Spalart–Allmaras approach is becoming the standard one–equation model in industrial use today.

Another commonly used class of turbulence models are the two–equation models, where two partial differential equations are involved in finding the turbulent viscosity. The most famous of the two equation models is probably the k-ϵ model of Launder and Sharma [48], but other variants such as the k-ω model [89, 104] and the SST model [65] are also in use. Wilcox [103] argues that two–equation models are superior to one–equation models in both accuracy and applicability and that one–equation models only show marginal improvement compared to algebraic models. In more recent publications [4, 27], it is reported that one–equation models are quite competitive to their two–equation counterparts. It is

even stated [4] that the overall performance of the Spalart–Allmaras model on relevant aerodynamic applications is better than the k-ϵ and k-ω models, although slightly inferior to the SST model. A disadvantage of two–equation models is that they typically require more calculation points in the boundary layers than the algebraic and one–equation models.

In conclusion, it can safely be said that there is no approach in turbulence modeling that is proven to be more accurate over a wide range of flows than its competitors and that turbulence modeling still can be considered an unsolved problem. While two–equation models were dominant in the 1980's, the general recent trend seems to have moved towards one–equation models, perhaps more a result of numerical convenience than improved accuracy. This trend is followed in this work, which concentrates on the use of the Spalart–Allmaras model.

3.3.1 Favre Averaging

The philosophy of full turbulence modeling is to time–average the governing equations in a way that suppresses the instantaneous fluctuations in the flowfield caused by turbulence, while still being able to capture time–dependency in the time scales of interest. Of course, this averaging procedure breaks down if the time scale of the physical phenomena of relevance is of similar time scale as that of the turbulence itself. This problem, for example, occurs in flutter applications where the turbulent fluctuations are the driving force. In such cases other averaging techniques, or alternatively LES, must be used. In many problems, the assumption is valid however, particularly for flows where a statistically averaged steady state solution is sought. For compressible flow, the most commonly used averaging procedure is that of Favre [20]. This procedure is density weighted and relies on Morkovins hypoproject [69] which states that the effect of density fluctuations on the turbulence is small, provided the fluctuations are small relative to the mean density. This assumption is usually valid for Mach numbers up to at least five [103].

To describe the Favre averaging procedure, it is convenient to first introduce the Reynolds time–averaging, defined by the formula

$$\overline{U} = \frac{1}{\Delta t}\int_{t}^{t+\Delta t} U(\boldsymbol{x}, s)\mathrm{d}s \tag{3.32}$$

for a general variable U. In this equation, Δt denotes the time interval over which the variable is averaged, and this should be large enough to smooth out the small turbulence mode oscillations. It is now easy to define the Favre average of a variable U as

$$\tilde{U} = \frac{\overline{\rho U}}{\overline{\rho}}, \tag{3.33}$$

which is a density–averaged version of the Reynolds averaging procedure. Using these definitions, one can define the fluctuating parts of an unknown as

$$U' = U - \overline{U} \tag{3.34}$$

$$U'' = U - \tilde{U} \tag{3.35}$$

for Reynolds and Favre averaging respectively. The expressions

$$\overline{U'} = 0 \tag{3.36}$$
$$\overline{\rho U''} = 0 \tag{3.37}$$
$$\overline{\rho U V} = \overline{\rho}\tilde{U}\tilde{V} + \overline{\rho U'' V''} \tag{3.38}$$

then follow. In the averaging procedure, it is found convenient to define the specific enthalpy as an independent variable, and to set

$$h = c_v T + \frac{p}{\rho} = \tilde{h} + h''. \tag{3.39}$$

Also, it is assumed that the fluctuating part of the viscosity is small, so that the viscosity coefficient can be considered constant throughout the averaging procedure. By Reynolds averaging the equation system (3.1), the set of equations

$$\int_\Omega \frac{\partial \hat{U}_i}{\partial t} \mathrm{d}\boldsymbol{x} + \int_{\partial\Omega} \hat{F}_{ij} n_j \mathrm{d}\boldsymbol{x} = \int_{\partial\Omega} \left(\hat{G}_{ij} + \hat{H}_{ij}\right) n_j \mathrm{d}\boldsymbol{x}, \tag{3.40}$$

results where

$$\hat{\boldsymbol{U}} = \begin{bmatrix} \overline{\rho} \\ \overline{\rho}\tilde{u}_1 \\ \overline{\rho}\tilde{u}_2 \\ \overline{\rho}\tilde{u}_3 \\ \overline{\rho}\tilde{\epsilon} \end{bmatrix}, \tag{3.41}$$

$$\hat{\boldsymbol{F}}_j = \begin{bmatrix} \overline{\rho}\tilde{u}_j \\ \overline{\rho}\tilde{u}_1\tilde{u}_j + \overline{p}\delta_{1j} \\ \overline{\rho}\tilde{u}_2\tilde{u}_j + \overline{p}\delta_{2j} \\ \overline{\rho}\tilde{u}_3\tilde{u}_j + \overline{p}\delta_{3j} \\ \tilde{u}_j\left(\overline{\rho}\tilde{\epsilon} + \overline{p}\right) \end{bmatrix} \tag{3.42}$$

and

$$\hat{\boldsymbol{G}}_j = \begin{bmatrix} 0 \\ \tilde{\tau}_{1j} \\ \tilde{\tau}_{2j} \\ \tilde{\tau}_{3j} \\ \tilde{u}_k\tilde{\tau}_{kj} - \tilde{q}_j \end{bmatrix}. \tag{3.43}$$

The correlation term tensor is given by

$$\hat{\boldsymbol{H}}_j = \begin{bmatrix} 0 \\ -\overline{\rho u_1'' u_j''} \\ -\overline{\rho u_2'' u_j''} \\ -\overline{\rho u_3'' u_j''} \\ -\overline{\rho u_j'' h''} + \overline{u_k'' \tau_{jk}} - \overline{\rho u_j'' \frac{1}{2} u_k'' u_k''} - \tilde{u}_k \overline{\rho u_k'' u_j''} \end{bmatrix} \tag{3.44}$$

and the averaged specific total energy becomes

$$\tilde{\epsilon} = c_v \tilde{T} + \frac{1}{2}\tilde{u}_k \tilde{u}_k + k \tag{3.45}$$

where

$$k = \frac{1}{2\overline{\rho}}\overline{\rho u_k'' u_k''} \tag{3.46}$$

is the specific turbulent kinetic energy. The averaged equation of state can be written as

$$\overline{p} = \overline{\rho} R \tilde{T}. \tag{3.47}$$

Equation (3.40) is called the *open* form of the Favre–average Navier–Stokes equations. The tensor $\hat{H}_{ij}$ represents the correlation terms resulting from the averaging procedure. It can be seen that while the form of the averaged continuity equation is unchanged compared to the original equation, the momentum equations have produced an additional term as a result of the triple correlation of the convective part. This term is referred to as the Favre–averaged Reynolds–stress tensor, denoted by

$$\tau_{ij}^R = -\overline{\rho u_i'' u_j''}. \tag{3.48}$$

From equation (3.44) it is seen that the energy equation has increased considerably in complexity, receiving several new terms from the averaging procedure. For the first correlation term, it is common to set

$$q_j^R = \overline{\rho u_j'' h''}, \tag{3.49}$$

and this is termed the turbulent heat flux, in analogy with the above definition of the Reynolds–stress tensor.

3.3.2 Closure Approximations

By applying the Favre averaging procedure, several unknown correlation terms have resulted. These have to be approximated, or modeled in some way, to close the system of equations. One common closure approximation is to assume that turbulence effects can be treated as diffusive effects in the original set of instantaneous Navier–Stokes equations. This Boussinesq assumption is written

$$\tau_{ij}^R = -\frac{2}{3}\mu_t \frac{\partial \tilde{u}_k}{\partial x_k}\delta_{ij} + \mu_t \left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i} \right) - \frac{2}{3}\overline{\rho} k \delta_{ij} \tag{3.50}$$

where a turbulent viscosity μ_t is introduced and the last term is included to guarantee the relation

$$\tau_{ii}^R = -2\overline{\rho} k. \tag{3.51}$$

The analogy of Reynolds [91] relates momentum and heat transfer and can be written

$$q_j^R = -\mu_t \frac{c_p}{Pr_t} \frac{\partial \tilde{T}}{\partial x_j} \tag{3.52}$$

where the turbulent Prandtl number is set to

$$Pr_t = 0.9 \tag{3.53}$$

in this work.

It is common, and valid for sub–hypersonic speeds [45, 103], to ignore the molecular diffusion and turbulent transport of turbulent kinetic energy appearing in equation (3.44), i.e. to set

$$\overline{u_j'' \tau_{ij}} \approx 0 \tag{3.54}$$

and

$$\overline{\rho u_j'' \frac{1}{2} u_k'' u_k''} \approx 0 \tag{3.55}$$

respectively. For the turbulence models used in this work, the kinetic turbulent energy term, k, is simply excluded from all the equations. This is a good approximation for sub–hypersonic flows [103] and results in the set

$$\int_\Omega \frac{\partial \hat{U}_i}{\partial t} \mathrm{d}\boldsymbol{x} + \int_{\partial\Omega} \hat{F}_{ij} n_j \mathrm{d}\boldsymbol{x} = \int_{\partial\Omega} \hat{G}^t_{ij} n_j \mathrm{d}\boldsymbol{x}, \tag{3.56}$$

of averaged equations where U_i and $\hat{F}_{ij}$ are given by (3.41) and (3.42), and

$$\hat{\boldsymbol{G}}^t_j = \begin{bmatrix} 0 \\ \tilde{\tau}^t_{1j} \\ \tilde{\tau}^t_{2j} \\ \tilde{\tau}^t_{3j} \\ \tilde{u}_k \tilde{\tau}_{kj} - \tilde{q}^t_j \end{bmatrix}. \tag{3.57}$$

Here

$$\tilde{\tau}^t_{ij} = \tilde{\tau}_{ij} + \tau^R_{ij} \tag{3.58}$$

where, as mentioned previously, the kinetic turbulence energy term is ignored. Also

$$\tilde{q}^t_j = \tilde{q}_j + q^R_j. \tag{3.59}$$

The similarity between equations (3.1) and (3.56) is obvious. The set of equations (3.56), (3.47), (3.45) is under–determined as the turbulent viscosity, μ_t, is unknown. The specific dependency of this variable with respect to the rest of the unknown field has to be found by the application of a turbulence model.

3.3.3 The Spalart–Allmaras turbulence model

The one–equation model of Spalart and Allmaras [88] was constructed using 'empiricism, arguments of dimensional analysis, Galilean invariance and selective dependence on molecular viscosity'. It was constructed for aerodynamic applications and is in heavy use in both industry and research communities [63, 66].

In this model, the kinematic turbulent viscosity is related to an eddy viscosity variable, $\tilde{\nu}$, through the definition

$$\nu_t = \frac{\mu_t}{\overline{\rho}} = \tilde{\nu} f_{v1}, \tag{3.60}$$

where the function

$$f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3} \tag{3.61}$$

is introduced. Here χ is defined as

$$\chi = \frac{\tilde{\nu}}{\nu} \tag{3.62}$$

and c_{v1} is a constant defined in Table 3.1. The eddy viscosity variable is found by solving the equation

$$\begin{aligned} \frac{\partial \tilde{\nu}}{\partial t} + \tilde{u}_j \frac{\partial \tilde{\nu}}{\partial x_j} &= c_{b1}\left[1 - f_{t2}\right]\tilde{S}\tilde{\nu} + \frac{1}{\sigma}\left[\frac{\partial}{\partial x_j}\left(\left(\nu + \tilde{\nu}\right)\frac{\partial \tilde{\nu}}{\partial x_j}\right) + c_{b2}\frac{\partial \tilde{\nu}}{\partial x_j}\frac{\partial \tilde{\nu}}{\partial x_j}\right] \\ &\quad - \left[c_{w1} f_w - \frac{c_{b1}}{\kappa^2} f_{t2}\right]\left[\frac{\tilde{\nu}}{d}\right]^2 + f_{t1}\Delta\tilde{u}^2. \end{aligned} \tag{3.63}$$

The left–hand side of the equation is the Lagrangian derivative of the eddy viscosity variable and the terms on the right–hand side represent production, diffusion, destruction and transition from left to right respectively. The additional terms introduced in equation (3.63) are defined by

$$\tilde{S} = \tilde{\omega} + \frac{\tilde{\nu}}{\kappa^2 d^2} f_{v2}, \tag{3.64}$$

$$f_{v2} = 1 - \frac{\chi}{1 + \chi f_{v1}}, \tag{3.65}$$

$$f_w = g\left[\frac{1 + c_{w3}^6}{g^6 + c_{w3}^6}\right]^{\frac{1}{6}}, \tag{3.66}$$

$$g = r + c_{w2}\left(r^6 - r\right), \tag{3.67}$$

$$r = \min\left(\frac{\tilde{\nu}}{\tilde{S}\kappa^2 d^2}, 10\right), \tag{3.68}$$

$$f_{t2} = c_{t3}\exp\left(-c_{t4}\chi^2\right), \tag{3.69}$$

$$f_{t1} = c_{t1} g_t \exp\left(-c_{t2}\frac{\tilde{\omega}_t^2}{\Delta\tilde{u}^2}\left[d^2 + g_t^2 d_t^2\right]\right), \tag{3.70}$$

$$g_t = \min\left(0.1, \frac{\Delta\tilde{u}}{\tilde{\omega}_t \Delta x}\right) \tag{3.71}$$

and the values of the constants appearing in the formulation are defined in Table 3.1. Also, d is the distance from a given point to the nearest wall and $\tilde{\omega}$ is the vorticity magnitude

$$\tilde{\omega} = \left(\left(\frac{\partial u_3}{\partial x_2} - \frac{\partial u_2}{\partial x_3}\right)^2 + \left(\frac{\partial u_1}{\partial x_3} - \frac{\partial u_3}{\partial x_1}\right)^2 + \left(\frac{\partial u_2}{\partial x_1} - \frac{\partial u_1}{\partial x_2}\right)^2\right)^{\frac{1}{2}}. \tag{3.72}$$

Table 3.1: Constants of the Spalart–Almaras turbulence model

$c_{b1} = 0.1355$	$c_{b2} = 0.622$
$\sigma = 2/3$	$\kappa = 0.41$
$c_{w1} = c_{b1}/\kappa^2 + (1 + c_{b2})/\sigma$	$c_{w2} = 0.3$
$c_{w3} = 2$	$c_{v1} = 7.1$
$c_{t1} = 1$	$c_{t2} = 2$
$c_{t3} = 1.1$	$c_{t4} = 2$

The function of the last term in (3.63) is to trigger turbulence in the flow. Points in 2D and curves in 3D need to be defined on the surfaces where transition to turbulence is to occur. The variable $\Delta\tilde{u}$ is the difference in velocity between the point and the associated trigger point, d_t denotes the closest distance to such a trigger point or curve, $\tilde{\omega}_t$ the vorticity magnitude and Δx the surface grid–spacing at this point.

For discretization purposes, it is convenient to rewrite the convective term in conservative form as

$$\tilde{u}_j \frac{\partial \tilde{\nu}}{\partial x_j} = \frac{\partial}{\partial x_j}\left(\tilde{u}_j \tilde{\nu}\right) - \tilde{\nu}\frac{\partial \tilde{u}_j}{\partial \tilde{x}_j}, \tag{3.73}$$

thus introducing an extra source term. This term is included here for completeness but is ignored by some authors [27] as the contribution of this term is usually small.

3.4 Boundary Conditions

For the equation systems (3.1), (3.26), (3.27), (3.28) or (3.56) to be well defined, boundary conditions have to be applied on the boundaries of the computational domain. The number and type of boundary conditions appropriate to a certain boundary depends upon which of the systems is considered and on the local Mach number. The number of boundary conditions required at a particular point is normally determined by the number of characteristics entering the domain. The number of boundary conditions required will therefore generally differ between inflow and outflow regions of the domain boundary.

3.4.1 Wall Boundary

For an inviscid wall, the relative normal component of the velocity must vanish to prevent fluid flowing through the surface. This requirement can be written

$$u_i n_i^w = u_i^w n_i^w \tag{3.74}$$

Table 3.2: Overview of the number of boundary conditions required at inflow boundaries in three dimensions.

Equation type	Supersonic	Subsonic
Inviscid	5	4
Viscous	5	5

where u_i^w and n_i^w are the wall velocity and normal vectors respectively. For viscous flow, the fluid particles are prescribed to stick to the wall, so that

$$u_i = u_i^w. \tag{3.75}$$

For a stationary boundary, this simply reduces to the requirement of zero velocity at the wall. In addition to these boundary conditions on the velocity components, a boundary condition must be applied to the temperature. Two common choices are either to assume an isothermal wall, by setting

$$T = T_w \tag{3.76}$$

at the wall boundary where T_w is a prescribed wall temperature or, alternatively, a Neumann boundary condition on the temperature can be applied assuming the wall to be isentropic

$$\frac{\partial T}{\partial n^w} \equiv \frac{\partial T}{\partial x_j} n_j^w = 0. \tag{3.77}$$

This assumption of an isentropic wall will be used in this work unless otherwise stated. The turbulent viscosity is set to zero at wall boundaries since there can be no velocity fluctuations at such points.

3.4.2 Inflow Boundary

For supersonic flow, five inflow boundary conditions are needed, i.e. fully determined flow. For subsonic inviscid flow, the number of conditions required is reduced to four since one characteristic is leaving the computational domain at this point. However this is not applicable to viscous flows where all five variables need to be specified, [24]. The number of boundary conditions required for these different inflow cases is summarized in Table 3.2.

In this work, the farfield inflow boundary conditions are imposed by using a characteristic approach [94]. For the Spalart-Allmaras turbulence model, $\tilde{\nu}$ is set to ten percent of the laminar viscosity at the inflow.

3.4.3 Outflow Boundary

The number of boundary conditions required at outflow boundaries is shown in Table 3.3, taken from [24]. It can be seen that the number of conditions required

Table 3.3: Overview of the number of boundary conditions required at outflow boundaries in three dimensions.

Equation type	Supersonic	Subsonic
Inviscid	0	1
Viscous	4	4

for outflow boundaries is greatly increased by the inclusion of viscous terms in the governing equations. For many viscous problems where the boundary of the computational domain is situated far enough from regions of significant gradients in the solution variables however, the inviscid boundary conditions can be applied instead. For problems where the boundary is situated in regions where gradients occur, such as for internal flow, the application of boundary conditions can be a significant challenge. For the external flows considered in this work, the method in [94] is also applied at outflow boundaries.

For external flow, the turbulence viscosity in the Spalart-Allmaras model is extrapolated from the internal values.

3.4.4 Symmetry Boundary

A symmetry boundary is defined in this work as a boundary where the normal velocity component with respect to the boundary is zero and the density and total energy have zero normal derivative, i.e.

$$u_j n_j^s = 0 \tag{3.78}$$

where n_j^s is the symmetry boundary normal and

$$\frac{\partial \rho}{\partial n^s} = \frac{\partial \epsilon}{\partial n^s} = 0. \tag{3.79}$$

Using symmetry boundaries is a convenient way of reducing the the problem size in computations with known symmetry planes, such as flow around symmetric aircraft configurations with no sideslip.

3.4.5 Engine Boundary

Certain testcases run in this work incorporate jet engines. The internal parts of these engines are considered to be outside the computational domain, thus introducing the need for boundary conditions at the engine inlets and outlets. For engine inlets, the mass flow, m_e, defined as

$$m_e = -\int_{\Gamma_e} \rho u_j n_j^e \mathrm{d}\boldsymbol{x} \tag{3.80}$$

is prescribed. Here Γ_e is the engine inlet boundary surface and n_j^e is the unit normal on the surface pointing into the computational domain. Engine outlets are

considered as supersonic inflow boundaries of the domain, i.e. fully determined. The turbulent viscosity of the Spalart–Allmaras turbulence model is prescribed a fixed value at engine outlets.

Chapter 4

Discretization Procedures

4.1 Introduction

This chapter considers the construction of a finite set of discrete equations to approximate the governing equations of fluid flow. Here, this construction is performed by an edge–based, node–centered finite volume approach. A short discussion comparing this method with alternative discretization approaches is presented. The method is formulated for unstructured meshes where hybrid elements are introduced in boundary layers for viscous flows. The dual mesh definitions of these meshes are described in detail. The approaches used for inviscid, viscous and time–accurate calculations are treated in separate sections. These sections describe the discretization of the various terms appearing in the different formulations, as well as the introduction of stabilizing dissipation. For time–accurate flow, a geometric conservative discretization approach for unstructured hybrid meshes on flows involving moving geometries is proposed.

4.2 Weighted Residual Methods

Most numerical schemes in use today can be considered a member of the *Weighted Residual Method* (WRM) family of discretization procedures. A system of n differential equations of the form

$$L_i(\boldsymbol{u}(\boldsymbol{x}), \boldsymbol{x}) = 0, \quad \boldsymbol{x} \in \Omega, \quad i \in [1, n] \tag{4.1}$$

is considered, where $\boldsymbol{u}$ and $\boldsymbol{x}$ are the dependent and independent variables respectively. A system of *weak forms* is introduced by integrating the equations over the numerical or discrete domain, Ω^D, in the manner

$$\int_{\Omega^D} L_i(\boldsymbol{u}(\boldsymbol{x}), \boldsymbol{x}) W_K(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} = 0 \tag{4.2}$$

where $\{W_K(\boldsymbol{x})\}, K \in \{1, N\}$ is a finite set of independent weighting functions. This introduces a set of $N \times n$ equations. The task is now to find an approximation of the dependent variables or unknowns, $\tilde{\boldsymbol{u}}$, that satisfy this same set of equations in addition to the boundary conditions imposed on the problem.

4.2.1 Finite Element Methods

By setting

$$\tilde{\boldsymbol{u}} = \sum_{P=1}^{N} \boldsymbol{u}_P S_P(\boldsymbol{x}), \tag{4.3}$$

where $\{S_P(\boldsymbol{x})\}, P \in \{1, N\}$ is a finite set of independent basis functions and $\{\boldsymbol{u}_P \in \mathbb{R}^n\}, P \in \{1, N\}$ is a set of constants, a system of equations with $N \times n$ unknowns has been created. Different selections of the basis and weighting functions results in different families of discretization schemes. Setting weight and basis functions to global trigonometric functions for example, results in spectral schemes, while applying the Dirac delta for the weighting functions yields finite difference schemes.

Finite element schemes are introduced by assuming local support for both weighting and basis functions. Typically, this is done by relating a discrete mesh to the numerical domain, where an assembly of non–overlapping elements covering the numerical domain is introduced, Figure 4.1. A set of N nodes is placed on separating surfaces between the elements, and in some cases inside the elements, at the coordinates $\{\boldsymbol{x}_P\}$. The grid lines connecting node pairs are termed edges or sides. The simplest and most commonly used finite element method is the Galerkin method where the weighting and basis functions are identical, $S_P = W_P, P = 1, N$, and satisfy the condition

$$S_P(\boldsymbol{x}_K) = \begin{cases} 1, & P = K \\ 0, & P \neq K. \end{cases} \tag{4.4}$$

For compressible fluid dynamics calculations, a commonly used finite element approach is based on a Galerkin scheme with linear basis functions [54, 76], but approaches with higher order elements are also in use [42, 105].

4.2.2 Finite Volume Methods

A finite volume scheme results by selecting the weighting functions in (4.2) to split the computational domain into a set of subdomains $\{\Omega_K \subset \Omega^D\}$

$$W_K(\boldsymbol{x}) = \begin{cases} 1, & \boldsymbol{x} \in \Omega_K \\ 0, & \boldsymbol{x} \notin \Omega_K. \end{cases} \tag{4.5}$$

The weighting functions are usually selected to be a partition of unity of the computational domain, i.e. with non–overlapping domains of influence and covering the entire computational domain. The definitions of these domains are usually related to meshes similar to the ones used in finite element schemes and such finite volume schemes are typically divided into two families; *node/vertex–based* and *element/cell–based*. The difference between these two families of schemes is found in the way the subdomain sets, or *control volumes*, $\{\Omega_K\}$ are constructed.

In node–based methods, the values of the discrete dependent variables are defined at the nodes, and the control–volumes are constructed around the nodes in a way such that only one node resides in each subdomain, the connecting

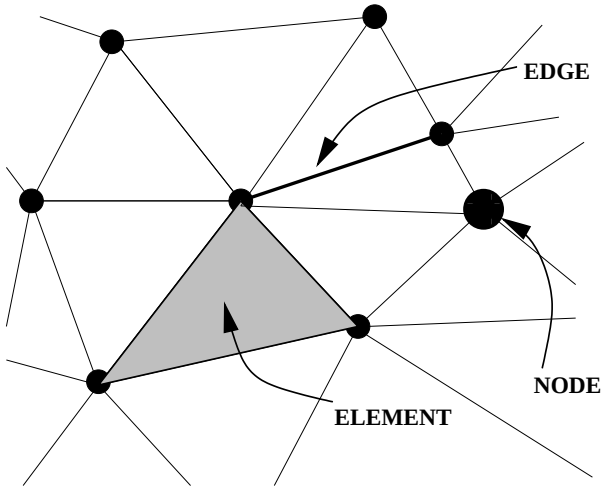

Figure 4.1: The nomenclature used for unstructured meshes, here illustrated by a two–dimensional triangular mesh.

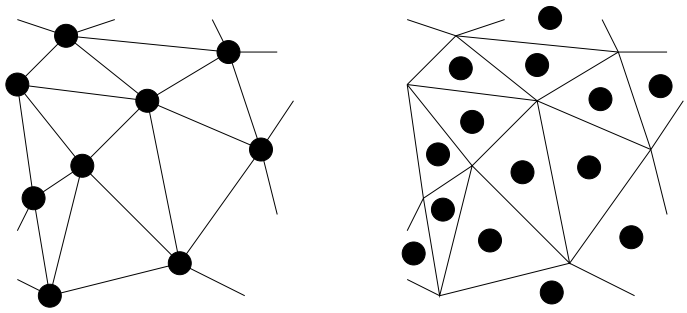

Figure 4.2: Illustration of node centered (left) and element centered (right) schemes. The filled circles illustrate the locations of the discrete unknowns.

surfaces between control–volumes is termed the *dual mesh*. An illustration of a node–based control–volume definition of an unstructured triangular mesh is shown in Figure 4.3.

Element–based methods do not require the creation of a dual mesh, but instead use the elements of the mesh themselves as control–volume definitions. The element–based schemes are usually categorized into two groups; the *node/vertex centered* where the unknowns are defined at the nodes of the mesh and *element/cell centered* where the unknowns are stored within the element, usually at the element centroid. An illustration of the two discretization types is shown in Figure 4.2.

It can be seen that the various formulations of the governing equations of fluid flow on integral form (3.1), (3.26), (3.27), (3.28), (3.56), essentially coincide with a finite volume definition of the weak form (4.2), where Gauss' theorem has been applied on the flux derivatives. The integral governing equations can thus be considered a weak form of a set of partial differential equations. The deduction of the integral governing equations is however more general compared to its differ-

ential counterpart, since less regularity is required for the variables. The surface and volume integrals appearing in the governing equations must be approximated numerically in some way to produce a set of discrete equations. These integrals are performed somewhat differently depending on the control volume definitions. The node–based schemes integrate the fluxes over the dual mesh, usually by looping over the edges of the original mesh. The element centered approach integrates over the edges (2D) or faces (3D) of the original mesh and the computational cost required is therefore proportional to the number of edges or faces in the mesh. The numerical integration for node centered schemes is usually performed as for the element centered approach and the values obtained are then redistributed to the nodes attached to the element.

The number of calculations required for a given grid on the different finite volume discretizations does not vary significantly on two–dimensional problems, since they are all proportional to the number of edges in the mesh. In three dimensions on tetrahedral meshes however, the number of element interfaces is greater than the number of edges in the mesh. While on hexahedral meshes, the computational effort is similar in the two schemes, for a three–dimensional tetrahedral mesh the ratio between edges and faces is approximately 6.5/11. In addition, the number of elements in tetrahedral meshes is usually of the order of 5.5 times larger than the number of nodes. Thus, storage requirements for node–based schemes will be less than that of an element–centered scheme for a simplex mesh. Another advantage of the node–based schemes is that they are similar, and in some cases equivalent to, well known discretizations of the finite difference and finite element types. For schemes where the unknowns are stored within the elements, there is also an added complication of applying boundary conditions, as the unknowns are not situated on the boundary of the computational domain.

4.3 Inviscid Scheme

The Euler equations are of hyperbolic type and convective in nature. This introduces problems of instability when traditional high order discretization schemes are applied. In the current scheme, a second order central difference type discretization is applied and stability is ensured by the explicit addition of a third order biharmonic term. The equations are formulated for use on simplex unstructured meshes, and the definition of the mesh dual is treated as well as a brief outline of the generation of meshes of this type.

4.3.1 Dual Mesh Construction

The dual mesh is constructed by connecting edge midpoints, element centroids, and in 3D, face–centroids in a way such that only one node is present in each control volume [95]. Each edge of the grid is associated with a segment of the dual mesh interface between the nodes connected to the edge. In two dimensions, this segment is constructed by connecting the centroids of the elements sharing the edge and the midpoint of the edge with straight lines. An illustration of such a dual mesh construction is shown in Figure 4.3, where the dual is denoted by dashed lines. The dual mesh interface inside the computational

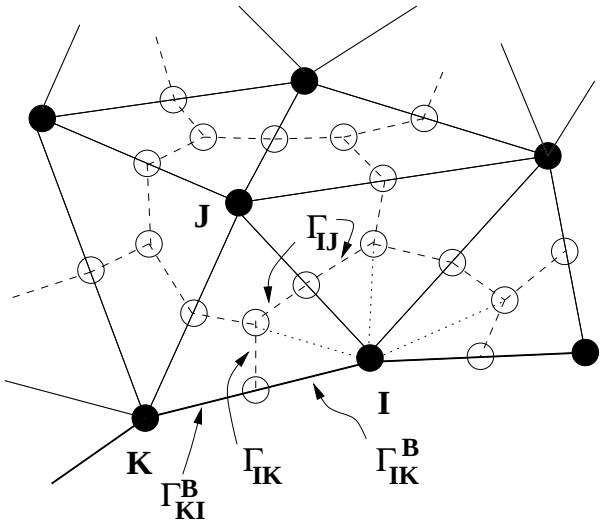

Figure 4.3: Illustration of the median dual (dashed lines) of a two–dimensional unstructured mesh.

domain surrounding node I is denoted Γ_I, while the parts of the dual situated on the computational boundary are termed Γ_I^B, so that $\Gamma_I \cap \Gamma_I^B = \emptyset$. The lines which define the control volume interface surrounding node I are denoted by Γ_I^K, so that $\Gamma_I \cup \Gamma_I^B = \bigcup_K \Gamma_I^K$. The subset of Γ_I associated with the edge spanning between nodes I and J, i.e. the lines touching the edge, is denoted Γ_{IJ}. For edges on the computational boundary, say edge $I-K$ in Figure 4.3, the internal dual has only one dual line associated with it, Γ_{IK}. However, such edges have in addition a dual segment on the computational boundary, i.e. Γ_{IK}^B, defined as the region on edge $I-K$ between node I and the edge midpoint.

For three dimensions the segment of the dual associated with an edge is a surface. This surface is defined using triangular facets, where each facet is connected to the midpoint of the edge, a neighbouring element centroid and the centroid of an element face connected to the edge. This is illustrated in Figure 4.4. The midpoint of the edge between node I and J is termed $\boldsymbol{x}_m^{IJ}$, the centroid of the face with vertices I, J and K is named $\boldsymbol{x}_s^{IJK}$ and the element centroid is designated by $\boldsymbol{x}_c$. The bold lines on the dual mesh in the figure illustrate the boundaries between the edges of which the dual mesh segment is associated. With this dual mesh definition, the control volume can be thought of as constructed by a set of tetrahedra with base on the dual mesh. Dual mesh intersection with the computational domain boundary is illustrated in Figure 4.5. The dual mesh associated with the boundary edge spanning between nodes I and J is defined as the two triangular facets with vertices at node I, the surface centroids and the midpoint of the edge. A complete dual around an internal node I is shown in Figure 4.6.

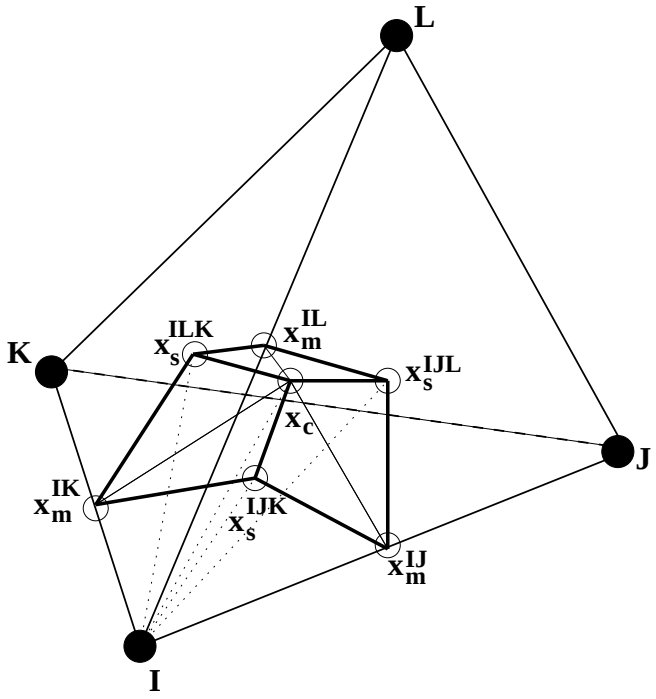

Figure 4.4: Illustration of the part of the dual mesh surrounding node I that is contained inside a tetrahedral element.

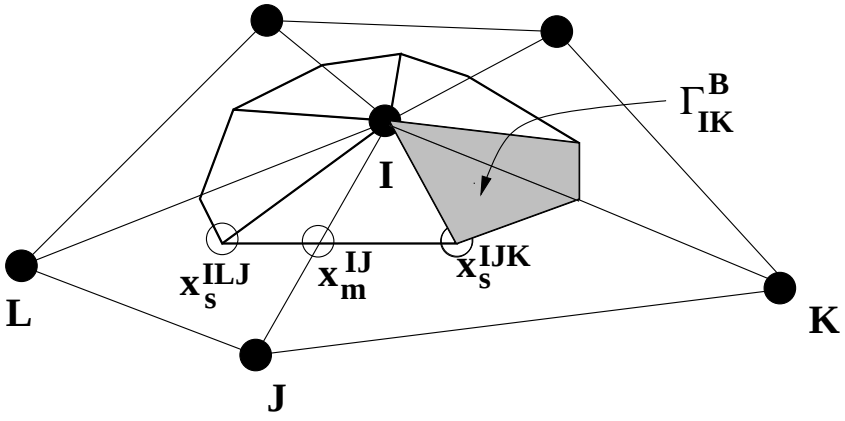

Figure 4.5: Illustration of the dual mesh construction on a triangular surface mesh in three dimensions.

4.3.2 Discretization of Inviscid Fluxes

To perform the numerical integration of the inviscid fluxes, a set of coefficients is calculated for each edge using the dual mesh segment associated with the edge. The values of these coefficients for an internal edge are given by the formula

$$C_j^{IJ} = \sum_{K \in \Gamma_{IJ}} A_{\Gamma_I^K} n_j^{\Gamma_I^K} \tag{4.6}$$

where $A_{\Gamma_I^K}$ is the area of facet Γ_I^K and $n_j^{\Gamma_I^K}$ is the outward unit normal vector of the facet from the viewpoint of node I, Figure 4.6. In a similar way, the contribution to boundary integrals from the computational boundary can be expressed using a coefficient

$$D_j^{IJ} = \sum_{K \in \Gamma_{IJ}^B} A_{\Gamma_I^K} n_j^{\Gamma_I^K}, \tag{4.7}$$

where Γ_{IJ}^B is the set of dual mesh faces on the computational boundary touching the edge between nodes I and J, Figure 4.5. In this case, $n_j^{\Gamma_I^K}$ denotes the normal of the facet in the outward direction of the computational domain.

The numerical integration of a flux over the dual mesh segment associated with an edge is then performed by assuming the flux to be constant, equal to its approximated value at the midpoint of the edge, over the segment, i.e. a form of midpoint quadrature. The calculation of a surface integral for the inviscid flux over the control volume surface for node I is then given by the formula

$$\int_{\partial\Omega_I} F_{ij} n_j \mathrm{d}\boldsymbol{x} \approx \sum_{J \in \Lambda_I} \frac{C_j^{IJ}}{2} \left(F_{ij}^I + F_{ij}^J\right) + \sum_{J \in \Lambda_I^B} D_j^{IJ} F_{ij}^I \tag{4.8}$$

where Λ_I denotes the set of nodes connected to node I by an edge and Λ_I^B denotes the set of nodes connected to node I by an edge on the computational boundary. The last term thus only contributes if I is a boundary node. The boundary terms are here treated in the classical finite volume way, using a local midpoint rule.

From equation (4.6) it can be seen that the internal edge coefficients are antisymmetric, i.e.

$$C_j^{JI} = -C_j^{IJ} \tag{4.9}$$

which means that the internal edge coefficients only need to be stored once for every edge. From equations (4.6) and (4.7) it follows that

$$\sum_{J \in \Lambda_I} C_j^{IJ} + \sum_{J \in \Lambda_I^B} D_j^{IJ} = 0 \tag{4.10}$$

so the scheme is consistent in the sense that the surface integral of a constant flux is numerically zero. These expressions guarantee the numerical scheme to be

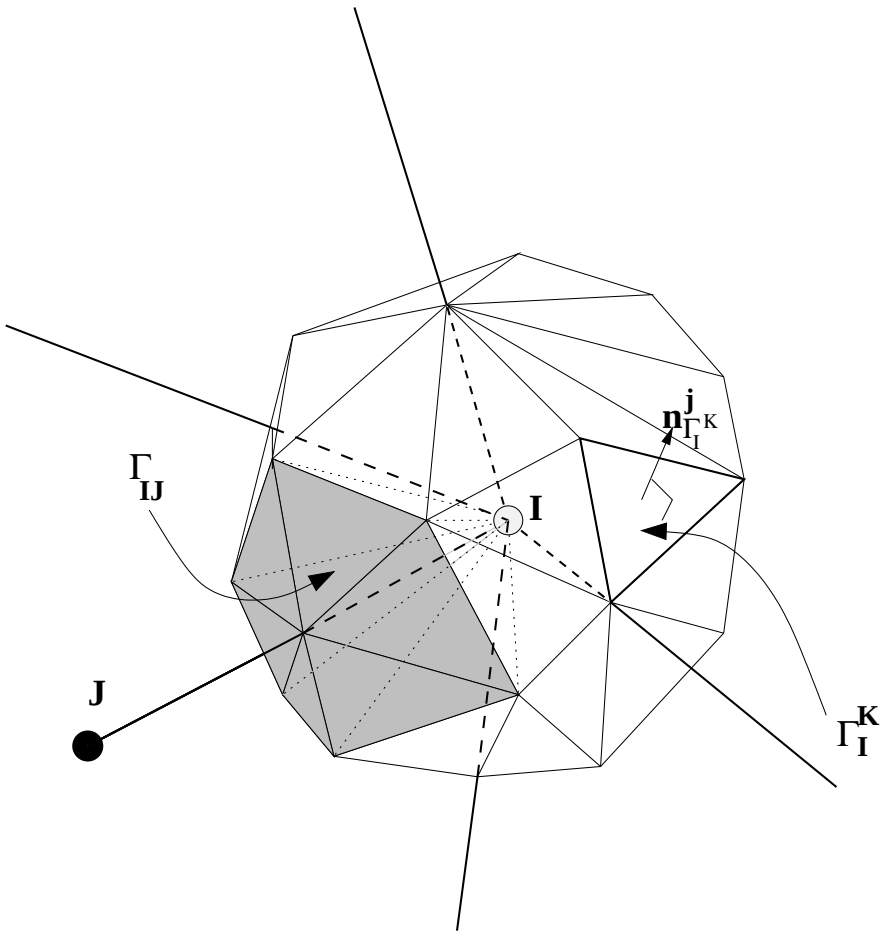

Figure 4.6: Illustration of the dual mesh surrounding an internal node I. The dual is constructed by a closed set of planar triangular facets $\left\{\Gamma_I^K\right\}$. Each facet only touches a single edge and the set of facets touching the edge spanning between nodes I and J is termed Γ_{IJ}. The internal edge coefficient of this edge is defined as a sum of the area times unit outward normal of the facets in Γ_{IJ}.

conservative, i.e. the sum of the numerical fluxes in the computational domain equals the flux over the computational domain boundary

$$\sum_I \left[\sum_{J \in \Lambda_I} \frac{C_j^{IJ}}{2} \left(F_{ij}^I + F_{ij}^J \right) + \sum_{J \in \Lambda_I^B} D_j^{IJ} F_{ij}^I \right] = \sum_I \sum_{J \in \Lambda_I^B} D_j^{IJ} F_{ij}^I. \tag{4.11}$$

The identity,

$$D_j^{IJ} = D_j^{JI} \tag{4.12}$$

for the boundary coefficients is also true as long as the edge midpoints are used in the dual mesh construction. This means that the boundary coefficients also need to be stored only once for each edge. Since the definitions of the internal and boundary edge coefficients are purely geometrical, they need only be calculated once, unless the mesh is deformed.

4.3.3 Artificial Dissipation

The numerical discretization described in section 4.3.2 is a typical central–difference approach, which is known to be unstable for hyberbolic–type equations. Over the history of CFD, stabilizing such schemes, while retaining second order accuracy, has been one of the main concerns [37]. It was soon found that addition of dissipation would make the scheme stable by damping the oscillations originating from the convection terms. The stabilizing dissipation can be thought of as adding a simplified discrete harmonic term to the discretized governing equations. This led to first order upwind schemes which are, although stable, too dissipative to produce accurate solutions for reasonable mesh sizes. The need for improved accuracy resulted in the discovery of several second order accurate approaches [7, 41, 58, 98]. One class of such schemes are the *artificial dissipation* schemes where a third order biharmonic operator is added to the equations. This term is however not capable of smoothing oscillations near high gradients of the flow, such as shocks, and requires a stabilizing local addition of harmonic dissipation which renders the scheme of first order locally. This scheme is related to TVD approaches [37] which can be viewed as special artificial dissipation schemes that guarantee the total variation of the dependent variables to be non–increasing with time.

The basic approach followed here is similar to that of the JST scheme [41], and introduces a third order biharmonic term of the form

$$H_i^I \equiv \sum_{J \in \Lambda_I} H_i^{IJ} = \sum_{J \in \Lambda_I} M_{ij}^{IJ} \left(E_j^J - E_j^I \right) \tag{4.13}$$

where

$$E_j^I = \sum_{K \in \Lambda_I} \left(U_j^K - U_j^I \right), \tag{4.14}$$

and H_i^{IJ} is termed the artificial dissipation flux. The term M_{ij}^{IJ} is the biharmonic dissipation matrix, which is $\mathcal{O}(h^{d-1})$ where d is the number of space dimensions

and h is a typical mesh spacing. For *scalar* dissipation schemes, M_{ij}^{IJ} is taken as diagonal while for *matrix* dissipation schemes, M_{ij}^{IJ} has non–diagonal entries.

It has been suggested [15] that a scaling can be applied to the artificial dissipation term to improve the stability of the scheme. If the inverse edge length is used as scaling coefficient, the scaled artificial dissipation flux takes the form

$$\hat{E}_j^I = \frac{1}{\sum_{K\in\Lambda_I} l_{IK}^{-1}} \sum_{K\in\Lambda_I} l_{IK}^{-1} \left(U_j^K - U_j^I\right) \tag{4.15}$$

where l_{IK} is the length of the edge between nodes I and K. This scaling results in an artificial dissipation scheme that is closer to discrete fourth order derivative formulae. For a stretched one–dimensional mesh, this scaled flux will eliminate a linear field exactly, which is not the case if the formula in equation (4.14) is employed.

If the dissipation matrix satisfies

$$M_{ij}^{IJ} = M_{ij}^{JI} \tag{4.16}$$

then

$$H_{ij}^{IJ} = -H_{ij}^{JI}, \tag{4.17}$$

and the corrected scheme retains the conservation property, (4.11). This also applies to the artificial dissipation scheme using equation (4.15).

As noted earlier, the biharmonic term is not capable of stabilizing the numerical scheme close to discontinuities in the flow and a harmonic term needs to be added for this purpose. This term can be written

$$Q_i^I = \sum_{J\in\Lambda_I} \epsilon_2 \alpha_{IJ} N_{ij}^{IJ} \left(U_j^J - U_j^I\right) \tag{4.18}$$

and is of first order. However, it only significantly contributes close to high pressure gradients due to the presence of a pressure switch

$$N_{ij}^{IJ} = \max\left(\left|\Delta p_I\right|, \left|\Delta p_J\right|\right) \delta_{ij} \tag{4.19}$$

where δ_{ij} is the Kronecker delta and

$$\Delta p_I = 12 \frac{\sum_{K\in\Lambda_I} \left(p_K - p_I\right)}{\sum_{K\in\Lambda_I} \left(p_K + p_I\right)}. \tag{4.20}$$

The constant ϵ_2 appearing in equation (4.18) is the harmonic dissipation factor which needs to be specified by the user. This parameter is usually given a value in the region $[0.1, 0.2]$. In this project, a value of 0.1 was used in all cases. The term α_{IJ} is defined as,

$$\alpha_{IJ} = \frac{1}{|\Lambda_I| + |\Lambda_J|} \min\left(\frac{V_I}{\Delta\tau_I}, \frac{V_J}{\Delta\tau_J}\right). \tag{4.21}$$

Here $|\Lambda_I|$ is the number of edges connected to node I, V_I is the volume (area in two dimensions) of the control volume containing node I and $\Delta\tau_I$ is the local time

step used in the explicit Runge–Kutta iterations of the solver, equation (5.44) to be described later. For inviscid flows, this time step is similar to

$$\Delta\tau \sim \frac{h}{|u|+c} \tag{4.22}$$

so that

$$\alpha_{IJ} \sim (|u|+c)\mathcal{O}(h^2) \tag{4.23}$$

in three dimensions. This can be recognized as the largest eigenvalue of the convective flux Jacobian times $\mathcal{O}(h^2)$. This scheme is thus similar to the JST scheme [41], where the largest eigenvalue is applied.

There are several choices available of the dissipation matrix M_{ij}^{IJ} for the biharmonic dissipation term. One alternative is to use the local Jacobian matrix of the inviscid flux

$$M_{ij}^{IJ} = \left|\frac{\partial F_{ik} r_k}{\partial U_j}\right|^{IJ}. \tag{4.24}$$

Here

$$r_k^{IJ} = \frac{1}{l_{IJ}}\left(x_k^J - x_k^I\right) \tag{4.25}$$

is the unit directional vector in the edge direction from node I to node J. The Jacobian is usually evaluated at the Roe state [82]. Alternatively, a diagonal dissipation matrix can be employed. The largest eigenvalue of the Jacobian matrix can be selected as the diagonal terms [41]. A modification of this approach is followed here where again the local time step is employed, such that

$$M_{ij}^{IJ} = m_{ij}^{IJ}\alpha_{IJ}, \tag{4.26}$$

where

$$m_{ij}^{IJ} = \max\left(0, \epsilon_4\delta_{ij} - \epsilon_2 N_{ij}^{IJ}\right). \tag{4.27}$$

Here the biharmonic dissipation factor, ϵ_4, is a user–specified parameter usually selected from the region $[0.1, 0.2]$. In this project, a value of 0.1 was used in all cases. It can be seen from equation (4.27) that the biharmonic term is eliminated in regions where the harmonic term is influential.

4.3.4 Discrete Equation

The discretization of the steady–state Euler equations (3.28) results in the equation system

$$\begin{aligned}\sum_{J\in\Lambda_I} \frac{C_j^{IJ}}{2}\left(F_{ij}^I + F_{ij}^J\right) + \sum_{J\in\Lambda_I^B} D_j^{IJ} F_{ij}^I \\ - \sum_{J\in\Lambda_I} m_{ij}^{IJ}\alpha_{IJ}\left(E_j^J - E_j^I\right) - \sum_{J\in\Lambda_I} \epsilon_2 \alpha_{IJ} N_{ij}^{IJ}\left(U_j^J - U_j^I\right) = 0\end{aligned} \tag{4.28}$$

where the first two terms represent the inviscid fluxes (section 4.3.2) and the last two terms the biharmonic and harmonic dissipation terms respectively (section 4.3.3).

4.4 Viscous Scheme

The viscous terms in the Navier–Stokes equations significantly complicate the discretization of the problem. High Reynolds–number flows incorporate boundary layers where the normal gradients are very large, which require dense meshes in this direction to accurately capture the flow features in such regions. Since the gradients are usually much smaller tangential to the boundary layer, a larger mesh spacing can be used in this direction. The result is highly stretched meshes in these regions of the flow, which can reduce the accuracy of the scheme. In this section, an investigation into the use of various schemes for discretizing the viscous fluxes and modifications to the artificial dissipation terms is presented. Hybrid meshes are introduced, where the boundary layer mesh consists of hexahedral or prismatic elements and the advantages of such meshes is discussed. For hybrid meshes, the generation of the dual mesh is somewhat more intricate than for simplex meshes and dependent on the manner in which the elements are created. A way of consistently defining this dual whilst ensuring its validity is proposed in this section.

4.4.1 Dual Mesh Construction

The median dual when used on the hybrid meshes generated by the procedure in section **??**, may produce control volumes that do not contain the associated node. This may occur locally in regions of high surface curvature or in the interface between the inner and outer meshes. To avoid this, the information contained in the original tetrahedral mesh is used. The quadrilateral faces of the merged elements are in general not planar and must be defined accordingly. By taking advantage of the information available from the simplex mesh, the quadrilateral element faces are defined as the union of the two triangular faces originating from the tetrahedral mesh. In this way, the surfaces of the elements are defined as closed collections of planar triangular facets which are valid due to the validity of the original tetrahedral mesh. The centroid of a quadrilateral face is defined as the midpoint of the edge shared between the two triangular facets making up the quadrilateral face. Centroids of triangular faces are taken as the nodal average since the triangles are considered planar. With such well defined element surfaces, it is now possible to define element centroids that ensure the validity of the dual mesh:

- For a pyramid element, the centroid is defined to be in the triangular merging face between the two tetrahedra making up the element, according to the formula

$$\boldsymbol{x}_m = \frac{1}{5}\left(2\boldsymbol{x}_1 + 2\boldsymbol{x}_3 + \boldsymbol{x}_5\right), \tag{4.29}$$

  where the local node indexes $1, 3$ and 5 denote the nodes on the diagonal of the quadrilateral base and pyramid top respectively.

- The centroid of a prism element is defined as the nodal average of the midpoints of the quadrilateral face centroids in the element.

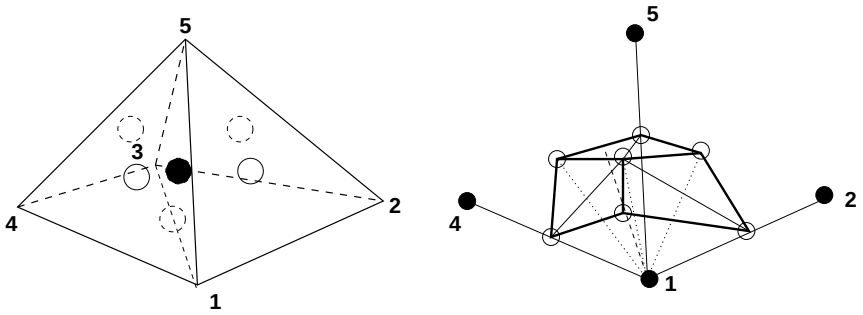

Figure 4.7: Illustration of the construction of the dual mesh for a pyramidal element.

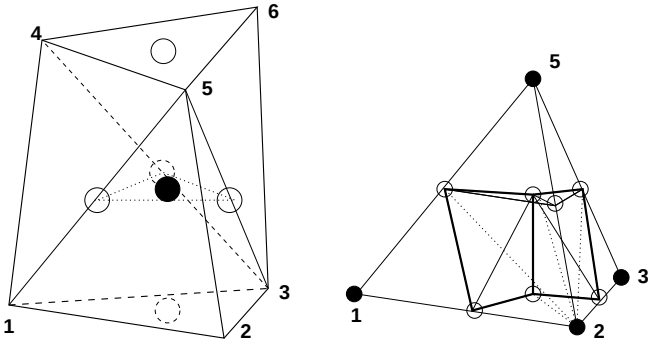

Figure 4.8: Illustration of the construction of the dual mesh for a prismatic element.

- The centroid of a hexahedral element is defined as the midpoint of the diagonal edge of the common quadrilateral face of the two prisms making up the element.

Illustration of the construction of the dual is shown in Figures 4.7, 4.8 and 4.9 for pyramids, prisms and hexahedra respectively. In the figures, the left illustration shows the position of the face centroids as open circles and the element centroid as a filled circle. The illustrations on the right show the part of the dual mesh surrounding a node that is contained within the element. In two dimensions, the centroid of a quadrilateral element is taken as the midpoint of the diagonal originating from the triangular mesh. These definitions of the dual mesh coincide with the median dual mesh for regular elements.

While the dual mesh construction for simplex meshes described in section 4.3.1 yields internal edge coefficients equivalent to an edge–based finite element method, the current approach differs significantly on hybrid meshes. This is easily realized by considering a quadrilateral 2D element. For a finite volume scheme, there will be no connection between diagonal nodes of the element since there are no edges connecting them. On the other hand, in finite element schemes, the bilinear basis functions associated with the nodes across the diagonals will interact and these nodes will influence each other directly in the scheme. It is possible to introduce artificial edges along the diagonals to obtain edge–based

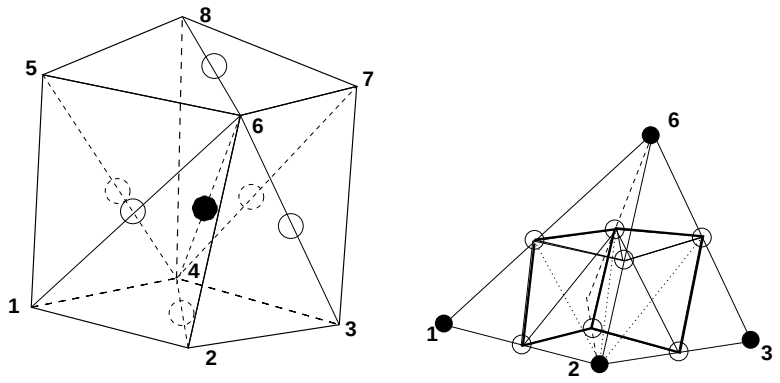

Figure 4.9: Illustration of the construction of the dual mesh for a hexahedral element.

finite element formulations for hybrid meshes, but here this added complexity is eliminated by using the finite volume approach.

4.4.2 Discretization of Viscous Fluxes

The viscous terms involve boundary integrals of gradients, which can be treated in several ways within the finite volume framework. The most straightforward way of computing these terms is first to calculate the derivatives of the velocity fields at every node using the formula

$$\int_{\Omega_I} \frac{\partial u_i}{\partial x_j} \mathrm{d}\boldsymbol{x} = \int_{\partial\Omega_I} u_i n_j \mathrm{d}\boldsymbol{x} \tag{4.30}$$

which in discrete form becomes

$$\left.\frac{\partial u_i}{\partial x_j}\right|^I \approx \partial_j^h u_i^I \equiv \frac{1}{V_I}\left[\sum_{J\in\Lambda_I} \frac{C_j^{IJ}}{2}\left(u_i^I + u_i^J\right) + \sum_{J\in\Lambda_I^B} D_j^{IJ} u_i^I\right]. \tag{4.31}$$

Here, the midpoint rule has been applied for the volume integral which is equivalent to lumping the mass matrix in finite element methods. The boundary integral of the gradient is now evaluated directly as

$$\begin{aligned}\int_{\partial\Omega_I} \frac{\partial u_i}{\partial x_j} n_k \mathrm{d}\boldsymbol{x} \;&\approx\; \sum_{J\in\Lambda_I} \frac{C_k^{IJ}}{2}\left(\partial_j^h u_i^I + \partial_j^h u_i^J\right) \\ &+\; \sum_{J\in\Lambda_I^B} D_k^{IJ}\partial_j^h u_i^I.\end{aligned} \tag{4.32}$$

This results in a second order accurate discretization on simplex meshes satisfying minimal criteria and smooth structured meshes, but the computational stencil stretches over five points in each direction. For a uniform structured quadrilateral or hexahedral mesh, this means that the nodes are decoupled from the closest neighbouring nodes in this term. In extreme conditions, this can induce checkerboarding in the solution field. In practice, this has not shown itself to be a

problem in the current approach as the decoupling is effectively hindered by the stabilizing biharmonic diffusion terms (section 4.3.3). This discretization does however in a sense double the actual grid–spacing used for the viscous terms, and it is usually desirable, from a discretization point of view, to employ schemes that are as compact as possible.

It is possible [90] to construct compact schemes for the viscous terms by evaluating the first order derivative at the edge midpoints using the formula (4.31) over a localized control volume for these points. The problem with such schemes is that they require construction of a new set of numerical coefficients. In addition, the computation of the boundary integrals become more complicated in the sense that the contribution of the integral associated with the edge becomes dependent on all of the nodes in the elements sharing the edge. Such an approach is therefore inherently non edge–based for hybrid meshes.

There is however an alternative way of discretizing the viscous terms that is edge–based and employs a more compact stencil, [16]. This is performed by splitting the evaluation of the derivative at an edge midpoint

$$\partial_j^h u_i^{IJ} = \frac{1}{2}\left(\partial_j^h u_i^I + \partial_j^h u_i^J\right) \tag{4.33}$$

into two components, tangential and normal to the edge,

$$\partial_j^h u_i^{IJ} = \partial_j^h u_i^{IJ}\Big|_t + \partial_j^h u_i^{IJ}\Big|_n \tag{4.34}$$

where

$$\partial_j^h u_i^{IJ}\Big|_t = r_k^{IJ}\partial_k^h u_i^{IJ} r_j^{IJ} \tag{4.35}$$

is the component in the direction of the edge and

$$\partial_j^h u_i^{IJ}\Big|_n = \partial_j^h u_i^{IJ} - r_k^{IJ}\partial_k^h u_i^{IJ} r_j^{IJ} \tag{4.36}$$

is the normal component. The unit vector in the edge direction, r_j^{IJ}, is defined in equation (4.25). The component of the derivative along the edge is then replaced by a finite difference approximation

$$\hat{\partial}_j^h u_i^{IJ}\Big|_t = \frac{1}{l_{IJ}}\left(u_i^J - u_i^I\right) r_j^{IJ} \tag{4.37}$$

which is second order accurate at the edge midpoint but only requires two nodes in its evaluation, as compared to four for the scheme in (4.35). The scheme

$$\int_{\partial\Omega_I} \frac{\partial u_i}{\partial x_j} n_k \mathrm{d}\boldsymbol{x} \approx \sum_{J\in\Lambda_I} C_k^{IJ}\left(\partial_j^h u_i^{IJ}\Big|_n + \hat{\partial}_j^h u_i^{IJ}\Big|_t\right) + \sum_{J\in\Lambda_I^B} D_k^{IJ}\partial_j^h u_i^I \tag{4.38}$$

can thus be employed in the evaluation of the viscous fluxes, where the gradients along edges are evaluated with the compact finite difference scheme and the remaining components are calculated in the standard finite volume way.

The viscous terms are thus approximated by

$$\int_{\partial\Omega_I} G_{ij} n_j \mathrm{d}\boldsymbol{x} \approx \sum_{J\in\Lambda_I} C_j^{IJ} G_{ij}^{IJ} + \sum_{J\in\Lambda_I^B} D_j^{IJ} G_{ij}^I \tag{4.39}$$

where

$$\boldsymbol{G}_j^{IJ} = \begin{bmatrix} 0 \\ \tau_{1j}^{IJ} \\ \tau_{2j}^{IJ} \\ \tau_{3j}^{IJ} \\ u_k^{IJ}\tau_{kj}^{IJ} - q_j^{IJ} \end{bmatrix} \quad \text{and} \quad \boldsymbol{G}_j^I = \begin{bmatrix} 0 \\ \tau_{1j}^{I} \\ \tau_{2j}^{I} \\ \tau_{3j}^{I} \\ u_k^{I}\tau_{kj}^{I} - q_j^{I} \end{bmatrix}. \tag{4.40}$$

Here, the discrete form of the deviatoric stress tensor at an edge midpoint is given by

$$\begin{aligned} \tau_{ij}^{IJ} = \frac{\mu_I + \mu_J}{2} \Bigg[-\frac{2}{3} \left(\partial_k^h u_k^{IJ}\Big|_n + \hat{\partial}_k^h u_k^{IJ}\Big|_t \right) \delta_{ij} \\ + \left(\partial_j^h u_i^{IJ}\Big|_n + \hat{\partial}_j^h u_i^{IJ}\Big|_t \right) + \left(\partial_i^h u_j^{IJ}\Big|_n + \hat{\partial}_i^h u_j^{IJ}\Big|_t \right) \Bigg], \end{aligned} \tag{4.41}$$

and the energy dissipation term is

$$u_k^{IJ}\tau_{kj}^{IJ} = \frac{u_k^I + u_k^J}{2}\tau_{kj}^{IJ}. \tag{4.42}$$

Similarly, the heat flux is calculated as

$$q_j^{IJ} = -\frac{k_I + k_J}{2} \left(\partial_j^h T_{IJ}\Big|_n + \hat{\partial}_j^h T_{IJ}\Big|_t \right). \tag{4.43}$$

For the boundary term, the point value of the deviatoric stress tensor is given by

$$\tau_{ij}^I = \mu_I \left(-\frac{2}{3}\partial_k^h u_k^I \delta_{ij} + \partial_j^h u_i^I + \partial_i^h u_j^I \right), \tag{4.44}$$

and the boundary term for the heat flux is

$$q_j^I = -k_I \partial_j^h T_I. \tag{4.45}$$

The viscous terms contribute mostly in the boundary layers which are characterized by high gradients normal to the wall and, except for localized regions such as around shocks, relatively small gradients tangential to the wall. The introduction of quasi–regular quadrilateral/hexahedral meshes with grid lines parallel and normal to the wall ensures that the high gradients are captured by the compact stencil and that the five point stencil terms are marginalized. The use of these meshes thus essentially reduces the current treatment of the viscous terms to the well known finite difference three–point scheme used for structured meshes. A one–dimensional analysis of the accuracy of an equivalent discretization has been performed [90], and it is concluded that the inviscid and viscous fluxes are second order accurate if the grid stretching coefficient, defined as

$$\beta = \frac{x_{I+1} - x_I}{x_I - x_{I-1}} \tag{4.46}$$

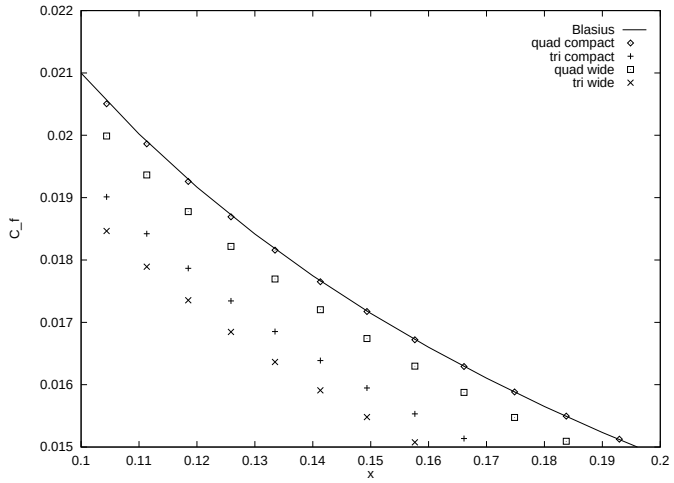

Figure 4.10: Comparison of the variation of the skin friction coefficient in flat plate flow, using three–point and five–point stencils for the viscous terms on triangular and quadratic structured meshes.

can be written

$$\beta = 1 + \mathcal{O}(h). \tag{4.47}$$

However, this is seldom true, as the grid stretching coefficient is usually taken to be $\beta = 1.2$ and there are therefore regions where the grid stretching reduces the accuracy of the scheme below second order.

A comparison between the skin friction coefficient calculated using the two viscous discretization schemes on a two–dimensional laminar flat plate flow is shown in Figure 4.10. The Reynolds number is ten thousand and the wall spacing of the first layer is equal to 10^{-5}. For this wall spacing it can be observed that the compact stencil produces superior results compared to those of the wider stencil, both for triangular and quadratic meshes.

4.4.3 Discretization of the Turbulence Model

The Spalart–Allmaras model requires the solution of a partial differential equation and this equation is discretized in a similar way to the governing equations. However, to avoid instabilities from the convective term, a first order upwind discretization of this term is used. This approach was also used in the original paper describing the turbulence model [88]. The discretization of the convective term is thus of the form

$$\begin{aligned}\int_{\partial\Omega_I} \tilde{u}_j \tilde{\nu} n_j \mathrm{d}\boldsymbol{x} \approx \sum_{J\in\Lambda_I} \Bigg[& \frac{C_j^{IJ}}{2}\left(\tilde{u}_j^I \tilde{\nu}_I + \tilde{u}_j^J \tilde{\nu}_I\right) \\ & - \left| C_j^{IJ} \frac{\tilde{u}_j^J \tilde{\nu}_J - \tilde{u}_j^I \tilde{\nu}_I}{\tilde{\nu}_I - \tilde{\nu}_J} \right| (\tilde{\nu}_J - \tilde{\nu}_I) \Bigg] + \sum_{J\in\Lambda_I^B} D_j^{IJ} \tilde{u}_j^I \tilde{\nu}_I,\end{aligned} \tag{4.48}$$

the added term introduces sufficient dissipation for stability. The accuracy of the resulting discretization of the turbulence model is therefore reduced to first order. But it is doubtful whether the turbulence model itself can be considered consistent, let alone second order accurate, so this numerical convenience does probably not significantly degenerate the capabilities of the turbulence model. The volume integrals are calculated using the midpoint rule and the gradients appearing in the model are calculated by the method of equation (4.31). The second order diffusion term is calculated using the compact stencil of equation (4.38).

4.4.4 Linear Preserving Artificial Dissipation

For non–regular meshes, the biharmonic dissipation described in section 4.3.3 will in general not totally eliminate an imposed linear field. Dissipation models which have this ability are termed linear preserving since they preserve the ability of a fourth order derivative to eliminate linear fields. It has been shown [15, 29, 68] that such a feature is desirable for artificial dissipation schemes. This is especially noticeable for viscous testcases on highly stretched meshes, where the original artificial dissipation scheme, described in section 4.3.3, is prone to be over–dissipative. This quality can, however, be enforced by modifying the standard JST correction. The method described in [15], was chosen in the current implementation and is restated here.

Assuming a general linear smoothing operator, H_i^I, that is consistent, i.e.

$$H_i^I(\boldsymbol{C}) = 0, \quad \boldsymbol{C} \in \mathbb{R}^n \tag{4.49}$$

which is the case for the JST correction, a modified flux $\hat{H}_i^I$ is introduced, defined as

$$\hat{H}_i^I(\boldsymbol{U}) = H_i^I(\boldsymbol{U}) - \left(\partial_j^h U_i^I\right) H_j^I(\boldsymbol{x}), \tag{4.50}$$

where the operator ∂_i^h is the numerical gradient defined in equation (4.31). This operator is capable of differentiating a linear field exactly. By assuming the field $\boldsymbol{U}$ to be linear, it follows that $\partial_i^h U_j^I$ is constant and that

$$\hat{H}_i^I(\boldsymbol{U}) = H_i^I(\boldsymbol{U} - \boldsymbol{x}\nabla\boldsymbol{U}) = H_i^I(\boldsymbol{C}) = 0. \tag{4.51}$$

A comparison between the two artificial dissipation schemes is shown in Figure 4.11 for the flat plate flow testcase described in section 4.4.2. It can be seen that the linear preserving scheme produces slightly better results than the standard scheme described in section 4.3.3. A point to note however, is that the scheme in section 4.3.3 using the scaled fluxes of equation (4.15) is linearly preserving for a regular stretched quadrilateral/hexahedral mesh in the directions of the grid lines. Since for viscous problems the gradients are essentially normal to the walls, this means that there is little to gain from the more complex approach if the hybrid meshes described in section **??** are used. In general, the original scheme is therefore used for viscous problems if hybrid meshes are applied, since this approach requires considerably less calculations than its linear preserving counterpart. For simplex meshes, the linearly preserving scheme is used unless otherwise specified.

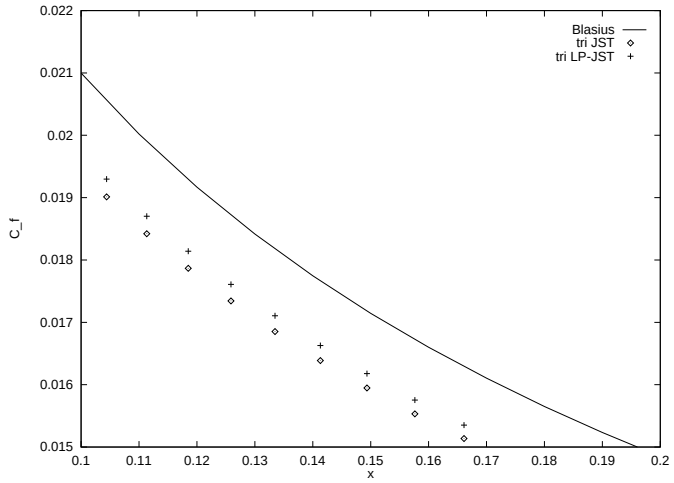

Figure 4.11: Comparison of skin friction coefficient on flat plate flow, using the artificial dissipation terms of sections 4.3.3 and 4.4.4.

4.4.5 Discrete Equation

The discrete system of the steady–state Navier–Stokes equations (3.27), can be written

$$\sum_{J\in\Lambda_I} \frac{C_j^{IJ}}{2}\left(F_{ij}^I + F_{ij}^J\right) + \sum_{J\in\Lambda_I^B} D_j^{IJ} F_{ij}^I - \sum_{J\in\Lambda_I} C_j^{IJ} G_{ij}^{IJ} - \sum_{J\in\Lambda_I^B} D_j^{IJ} G_{ij}^I$$
$$- \sum_{J\in\Lambda_I} M_{ij}^{IJ}\left(E_j^J - E_j^I\right) - \sum_{J\in\Lambda_I} \alpha_{IJ} N_{ij}^{IJ}\left(U_j^J - U_j^I\right) = 0, \quad (4.52)$$

where the first two terms represent the inviscid fluxes (section 4.3.2), the two next terms represent the viscous fluxes (section 4.4.2) and the last two terms introduce stabilizing numerical diffusion (sections 4.3.3 and 4.4.4). If turbulent flow is considered, the turbulent viscosity is found by solving the discrete equation for the Spalart–Allmaras model as explained in section 4.4.3.

4.5 Time–Accurate Scheme

Certain classes of problems in CFD are time–dependent by nature. Examples of such problems are maneuvering, store release and oscillations of wings or control surfaces on aircraft induced by freestream turbulence or flutter. All of these examples require time–accurate simulations to accurately capture the physics of the flow. The introduction of time–accuracy in a steady–state scheme is usually fairly straightforward for problems where there is no mesh movement. If the geometry itself changes with time however, there are more considerations to be made if accurate and stable schemes are to be found. In this section a scheme for solving such problems is proposed, and discretization approaches, as well as meshing issues, are discussed.

4.5.1 Time Discretization

For certain problems the geometry itself can be considered stationary while the fluid movement is unsteady. In these cases, the spatial discretization of the governing equations is unaffected and only the addition of the time term appearing in equations (3.1), (3.26) and 3.56) is required for time–accuracy. There is, in principle, no reason to treat the time dimension any differently from the spatial dimensions, and some authors apply the same discretization on all independent unknowns [92]. Due to the regularity of the time discretization however, it is quite common to discretize the time dependency independently by a simple finite difference scheme [8, 73, 97] and this approach is used here.

A commonly used time discretization is the two–level, first order backward Euler scheme,

$$\left.\frac{\partial U_i}{\partial t}\right|_{t=t_n} = \frac{1}{\Delta t}\left(U_i^n - U_i^{n-1}\right) + \mathcal{O}(\Delta t) \tag{4.53}$$

where Δt is the time increment, assumed to be constant in the calculation, and where the superscript denotes the discrete time level at which the term is to be evaluated.

For higher time–accuracy, the three–level, second order accurate backward difference scheme,

$$\left.\frac{\partial U_i}{\partial t}\right|_{t=t_n} = \frac{1}{\Delta t}\left(\frac{3}{2}U_i^n - 2U_i^{n-1} + \frac{1}{2}U_i^{n-2}\right) + \mathcal{O}(\Delta t^2) \tag{4.54}$$

can be used. While more accurate, this scheme introduces the need to store an additional set of unknowns. This addition to the memory requirements of the flow solver described here is not considered significant however.

4.5.2 Mesh Movement

If the geometry under consideration is moving with time, the mesh will need to change in accordance with this movement. One way of updating the mesh is to remesh the entire computational domain every time the geometry moves. This is not only computationally expensive but also results in reduced accuracy, due to interpolation errors. In addition, certain desirable features of the calculation, such as geometric conservation, described in section 4.5.4, are difficult to ensure if the mesh structure changes between time steps. It is, therefore, good practice to avoid remeshing whenever possible and, instead, strive to retain the connectivity of the previous mesh. For external flows, this is usually achieved by holding the outer mesh boundary fixed, moving the inner mesh boundary in accordance with the geometry movement and moving the internal mesh nodes to achieve the desired mesh quality. Several mesh movement approaches have been investigated by other researchers. One approach uses a stress/strain analogy where the computational domain is thought of as an elastic material subjected to a deformation. This approach involves solving a Laplace–type system [14]. The approach used here is assuming that the edges of the mesh are behaving like springs connecting the nodes together, [6]. The inner boundary nodes are then moved in small

increments, letting the nodes obtain internal equilibrium for each increment. This is done by solving for the system

$$\sum_{J \in \Lambda_I} s_{IJ} \left(l_{IJ} - l_{IJ}^{prev} \right) = 0, \tag{4.55}$$

where l_{IJ}^{prev} and l_{IJ} are the lengths of the edge between nodes I and J before and after the movement respectively, and s_{IJ} is the spring coefficient, which is usually taken as the inverse length of the edge. Typically 50 increments are used for the examples in this project. This approach is robust and fast, usually requiring about 10% of the total CPU–time required to solve the system time–accurately. For viscous calculations involving boundary layers, a specified number of the wall–layers closest to the wall are held fixed in respect to the surface, while the outer part of the layers are allowed to move in a normal manner. Although sufficient for the examples performed here, it is expected that this approach might not be sufficiently robust for complicated movement patterns. If this is the case, the torsional spring method [18] or the approach in [14] may be more suitable.

4.5.3 Remeshing

Sometimes it is impossible to avoid remeshing while still keeping the necessary mesh quality. An example of such a situation is store separation, where the geometry splits into several parts that move relative to each other. If the distance between two bodies is increasing, the mesh is likely to become too coarse in the movement direction. Applying mesh movement in this case will result in mesh stretching, which can effect accuracy. If two bodies are moving towards each other, the mesh will be squeezed between them, again resulting in anisotropic mesh spacing in such regions. The regions which are unsuitable for mesh movement are usually relatively small however, and this is taken advantage of in the current approach by applying localized remeshing. The regions which need remeshing are found by using a mesh quality indicator, which in the current implementation is the ratio of the volume of an element at the current time level to the element volume in the original mesh. The mesh is then removed in the regions defined by this process, creating one or several holes in the computational domain. Each of these holes are then meshed in a normal manner applying a Delaunay scheme using the hole surface as the surface triangulation. If the boundary surface can not be recovered, another element layer is removed from the original mesh and the new hole is remeshed [35].

The unknown fields in remeshed regions are transferred between meshes using a linear interpolation scheme. In addition, the coordinates of the nodes at the current time level are interpolated back to the hole at the previous time level. In this way, a new previous mesh level is constructed, with the connectivities of the current mesh level. This allows for a unified treatment of the remeshed regions and the regions where mesh movement is applied.

4.5.4 Arbitrary Lagrangian–Eulerian Schemes

For problems involving mesh deformation, the spatial discretization of the governing equations is affected. One way of solving such problems is to retain the

Eulerian form of the governing equations (3.1), (3.26), 3.56) but allowing the control volumes to move independently of the flow. This approach is hence termed Arbitrary Lagrangian–Eulerian (ALE), [17]. By allowing the control volumes to be time–dependent, the governing equation (3.1) can be written as

$$\int_{\Omega(t)} \frac{\partial U_i}{\partial t} \mathrm{d}\boldsymbol{x} + \int_{\partial\Omega(t)} F_{ij} n_j \mathrm{d}\boldsymbol{x} = \int_{\partial\Omega(t)} G_{ij} n_j \mathrm{d}\boldsymbol{x}. \tag{4.56}$$

By applying the generalized Leibnitz rule on the time term, the equation

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_{\Omega(t)} U_i \mathrm{d}\boldsymbol{x} + \int_{\partial\Omega(t)} \left(F_{ij} - v_j U_i\right) n_j \mathrm{d}\boldsymbol{x} = \int_{\partial\Omega(t)} G_{ij} n_j \mathrm{d}\boldsymbol{x} \tag{4.57}$$

results where v_j is the velocity of the control volume boundary. The Eulerian description of flows incorporating moving meshes is therefore achieved by the addition of an extra flux term, termed the ALE flux. The evaluation of the time derivative term becomes

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_{\Omega_I(t_n)} U_i \mathrm{d}\boldsymbol{x} \bigg|_{t=t_n} \approx \partial_h^t (V U_i)^{I,n} \equiv \frac{1}{\Delta t} \left(V_I^n U_i^{I,n} - V_I^{n-1} U_i^{I,n-1} \right), \tag{4.58}$$

for the first order scheme (4.53) and

$$\partial_h^t \left(V U_i\right)^{I,n} = \frac{1}{\Delta t} \left(\frac{3}{2} V_I^n U_i^{I,n} - 2 V_I^{n-1} U_i^{I,n-1} + \frac{1}{2} V_I^{n-2} U_i^{I,n-2} \right) \tag{4.59}$$

for the second order scheme of equation (4.54).

An important property of equation (4.57) is geometric conservation. Geometric conservation ensures that the control–volume movement itself has no direct effect on the fluxes, in the sense that if the unknown field is constant, the solution does not change in time, i.e.

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_{\Omega(t)} \mathrm{d}\boldsymbol{x} - \int_{\partial\Omega(t)} v_j n_j \mathrm{d}\boldsymbol{x} = 0. \tag{4.60}$$

It is desirable to retain this quality numerically, after discretization. This was first shown in [93] and since advocated in [47, 51]. For a numerical scheme, the discrete ALE term for node I at time level n is termed $P_i^{I,n}$, so that

$$P_i^{I,n} \approx \int_{\partial\Omega_I(t_n)} v_j^n U_i^n n_j^n \mathrm{d}\boldsymbol{x}. \tag{4.61}$$

The numerical integration of this term is performed in the usual way of summing the edge contributions. It is therefore convenient to introduce the numerical ALE coefficients S_{IJ}^n, T_{IJ}^n, for the edge between nodes I and J, such that

$$P_i^{I,n} = \sum_{J \in \Lambda_I} \frac{S_{IJ}^n}{2} \left(U_i^{I,n} + U_i^{J,n} \right) + \sum_{J \in \Lambda_I^B} T_{IJ}^n U_i^{I,n}. \tag{4.62}$$

The task of finding ALE coefficients that render the numerical scheme geometric conservative is not trivial and has been treated by several authors [47, 51, 73, 93].

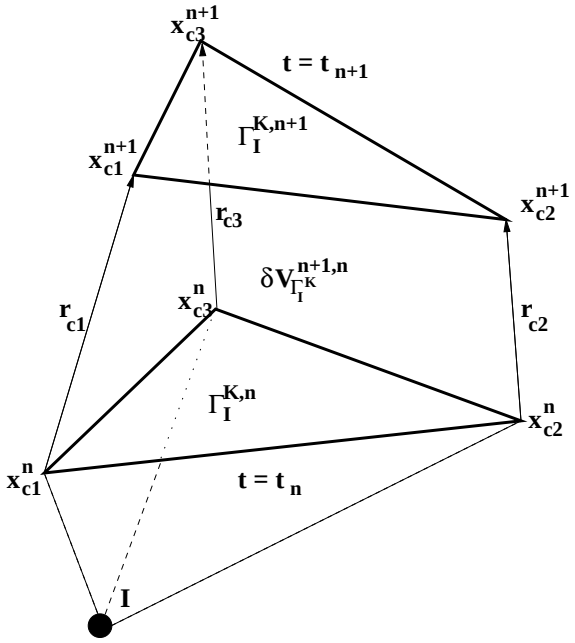

Figure 4.12: Illustration of the terminology used for the movement of a triangular facet of the dual mesh for node I.

Here the approach of [73] is followed. This was originally stated for simplex meshes, but is described here in a form suitable for the hybrid meshes introduced in section **??**, with the control volume definitions of section 4.4.1.

The key point in [73] is the assumption that each node in the mesh moves in a linear fashion between time levels n and $n+1$. Under this assumption, the movement of a triangular facet can be easily described. In the following, the notation used in section 4.3.2 is applied. An illustration of the dual mesh facet Γ_I^K is given in Figure 4.12. The points $\boldsymbol{x}_{c1}, \boldsymbol{x}_{c2}$ and $\boldsymbol{x}_{c3}$ are situated at the vertices of the facet and are moved in a linear fashion in space–time from time level n to time level $n+1$. The vertices may be element centroids, face centroids or edge midpoints for any kind of mesh with a dual mesh consisting of planar triangular facets.

An arbitrary point positioned on the facet plane

$$\boldsymbol{x}_p^n = (1-\alpha-\beta)\boldsymbol{x}_{c1} + \alpha\boldsymbol{x}_{c2} + \beta\boldsymbol{x}_{c3} \tag{4.63}$$

is considered. Since the nodes move in a linear fashion between time levels, the position of node $\boldsymbol{x}_{c\cdot}$ can be written

$$\boldsymbol{x}_{c\cdot}(t) = \boldsymbol{x}_{c\cdot}^n + (t-t_n)\left.\frac{\mathrm{d}\boldsymbol{x}_{c\cdot}}{\mathrm{d}t}\right|^n, \quad t \in (t_n, t_{n+1})\,. \tag{4.64}$$

By inserting this into equation (4.63), the expression

$$\boldsymbol{x}_p(t) = \boldsymbol{x}^n + (t-t_n)\left.\frac{\mathrm{d}\boldsymbol{x}_p}{\mathrm{d}t}\right|^n \tag{4.65}$$

follows. This means that the movement of all points in the facet plane is linear if the nodes move in a linear fashion. It is worth noticing that quadrilateral faces will not, in general, remain planar if subjected to linear node movement. This significantly complicates the issue of geometric conservation for dual meshes comprising of quadrilaterals, such as the standard median–dual for hybrid meshes.

The volume swept over by triangular facet Γ_I^K between time levels n and $n+1$ is therefore equal to the volume shown in Figure 4.12, which is given as

$$\delta V_{\Gamma_I^K}^{n+1,n} = \int_{t_n}^{t_{n+1}} \int_{\Gamma_I^K} v_j^{\Gamma_I^K} n_j^{\Gamma_I^K} \, \mathrm{d}\boldsymbol{x}\mathrm{d}t. \tag{4.66}$$

It can be shown, [51, 73], that

$$\delta V_{\Gamma_I^K}^{n+1,n} = \frac{1}{9}\left(A_{\Gamma_I^K}^{n+1} n_j^{\Gamma_I^K,n+1} + A_{\Gamma_I^K}^{n} n_j^{\Gamma_I^K,n} + A_{\Gamma_I^K}^{*} n_j^{\Gamma_I^K,*}\right)\left(r_j^{c1} + r_j^{c2} + r_j^{c3}\right), \tag{4.67}$$

where

$$\begin{aligned}
A_{\Gamma_I^K}^{n+1}\boldsymbol{n}_{\Gamma_I^K}^{n+1} &= \frac{1}{2}\left(\boldsymbol{x}_{c2}^{n+1} - \boldsymbol{x}_{c1}^{n+1}\right) \times \left(\boldsymbol{x}_{c3}^{n+1} - \boldsymbol{x}_{c1}^{n+1}\right) && (4.68)\\
A_{\Gamma_I^K}^{n}\boldsymbol{n}_{\Gamma_I^K}^{n} &= \frac{1}{2}\left(\boldsymbol{x}_{c2}^{n} - \boldsymbol{x}_{c1}^{n}\right) \times \left(\boldsymbol{x}_{c3}^{n} - \boldsymbol{x}_{c1}^{n}\right) && (4.69)\\
A_{\Gamma_I^K}^{*}\boldsymbol{n}_{\Gamma_I^K}^{*} &= \frac{1}{4}\left(\boldsymbol{x}_{c2}^{n+1} - \boldsymbol{x}_{c1}^{n+1}\right) \times \left(\boldsymbol{x}_{c3}^{n} - \boldsymbol{x}_{c1}^{n}\right) \\
&+ \frac{1}{4}\left(\boldsymbol{x}_{c2}^{n} - \boldsymbol{x}_{c1}^{n}\right) \times \left(\boldsymbol{x}_{c3}^{n+1} - \boldsymbol{x}_{c1}^{n+1}\right) && (4.70)
\end{aligned}$$

and

$$\begin{aligned}
\boldsymbol{r}_{c1} &= \boldsymbol{x}_{c1}^{n+1} - \boldsymbol{x}_{c1}^{n} && (4.71)\\
\boldsymbol{r}_{c2} &= \boldsymbol{x}_{c2}^{n+1} - \boldsymbol{x}_{c2}^{n} && (4.72)\\
\boldsymbol{r}_{c3} &= \boldsymbol{x}_{c3}^{n+1} - \boldsymbol{x}_{c3}^{n}. && (4.73)
\end{aligned}$$

A local geometric conservation law

$$\delta V_{\Gamma_I^K}^{n+1,n} = \Delta t v_j^{\Gamma_I^K,*} n_j^{\Gamma_I^K,*} A_{\Gamma_I^K}^{*}, \tag{4.74}$$

can now be applied to the facet, where $v_j^{\Gamma_I^K,*}$ is an averaged facet velocity, $A_{\Gamma_I^K}^{*}$ is the averaged facet area and $n_j^{\Gamma_I^K,*}$ is the average normal between time levels n and $n+1$. The right hand side of this equation can be recognized as the contribution of the numerical ALE term for a single facet, multiplied by the time step. Since the dual mesh definitions described in 4.4.1 are constructed by collections of triangular facets, the formulation of a geometric conservative scheme for a hybrid mesh is now straightforward. By setting

$$\Delta t S_{IJ}^{n+1} = \sum_{K \in \Gamma_{IJ}} \delta V_{\Gamma_I^K}^{n+1,n}, \tag{4.75}$$

and similarly for the computational boundary

$$\Delta t T_{IJ}^{n+1} = \sum_{K \in \Gamma_{IJ}^B} \delta V_{\Gamma_I^K}^{n+1,n}, \tag{4.76}$$

where again the notation used in section 4.3.2 has been used, a numerical ALE flux that satisfies a geometric conservation law is found. This can be seen by summing over the edges connected to a given node I,

$$\Delta t \sum_{J \in \Lambda_I} S_{IJ}^{n+1} + \Delta t \sum_{J \in \Lambda_I^B} T_{IJ}^{n+1} = \sum_{J \in \Lambda_I} \left[\sum_{K \in \Gamma_{IJ}} \delta V_{\Gamma_I^K}^{n+1,n} + \sum_{K \in \Gamma_{IJ}^B} \delta V_{\Gamma_I^K}^{n+1,n} \right] = V_I^{n+1} - V_I^n, \tag{4.77}$$

where the last equality must be true since the control volume is closed. This ensures that the numerical geometric conservation law is satisfied when the backward Euler time discretization of equation (4.58) is used.

If the second order time discretization given in (4.59) is applied however, a modified version of the ALE flux has to be employed to ensure numerical geometric conservation. Here, the modification

$$\hat{S}_{IJ}^{n+1} = \frac{3}{2} S_{IJ}^{n+1} - \frac{1}{2} S_{IJ}^n \tag{4.78}$$

$$\hat{T}_{IJ}^{n+1} = \frac{3}{2} T_{IJ}^{n+1} - \frac{1}{2} T_{IJ}^n \tag{4.79}$$

is used, [97], which makes the numerical ALE coefficient second order in time. By using equation (4.77), it is seen that

$$\frac{3}{2} V_I^{n+1} - 2 V_I^n + \frac{1}{2} V_I^{n-1} = \Delta t \sum_{J \in \Lambda_I} \hat{S}_{IJ}^n + \Delta t \sum_{J \in \Lambda_I^B} \hat{T}_{IJ}^n \tag{4.80}$$

which is a geometric conservation law valid for the three–step time discretization.

From the above discussion, it is clear that the internal ALE coefficients are anti–symmetric,

$$S_{IJ}^n = -S_{JI}^n, \tag{4.81}$$

since the volume swept over by each facet is the same for the two control volumes connected by an edge, but of opposite sign. This means that the internal ALE coefficients only need to be stored once for every edge. There is however no direct relationship between T_{IJ}^n and T_{JI}^n, and both of these coefficients need to be calculated and stored.

The boundary conditions of equations (3.74) or (3.75) are applied by using the formula

$$\boldsymbol{u}_I^{w,n+1} = \frac{1}{\Delta t} \left(\boldsymbol{x}_I^{n+1} - \boldsymbol{x}_I^n \right) \tag{4.82}$$

for the wall velocity if the first order scheme is used, and

$$\boldsymbol{u}_I^{w,n+1} = \frac{1}{\Delta t}\left(\frac{3}{2}\boldsymbol{x}_I^{n+1} - 2\boldsymbol{x}_I^{n} + \frac{1}{2}\boldsymbol{x}_I^{n-1}\right) \tag{4.83}$$

if the second order scheme is used. This is consistent with the dual mesh velocity definitions used in the two time discretization approaches.

Since for remeshed regions, the coordinates of the current time level are interpolated back to the previous time level, a new previous time level mesh with the current mesh connectivities is created. The mesh discretization is then performed in the same way as for the moved regions of the mesh. In the current implementation, the first order time discretization scheme is used if remeshing has been performed.

4.5.5 Discrete Equation

For time–accurate viscous flow using the second order time stepping scheme, the numerical system

$$\begin{aligned}
&\frac{1}{\Delta t}\left(\frac{3}{2}V_I^n U_i^{I,n} - 2V_I^{n-1}U_i^{I,n-1} + \frac{1}{2}V_I^{n-2}U_i^{I,n-2}\right) \\
&\quad + \sum_{J\in\Lambda_I}\frac{C_j^{IJ,n}}{2}\left(F_{ij}^{I,n} + F_{ij}^{J,n}\right) + \sum_{J\in\Lambda_I^B} D_j^{IJ,n}F_{ij}^{I,n} \\
&\quad - \sum_{J\in\Lambda_I}\frac{\hat{S}_{IJ}^n}{2}\left(U_i^{I,n} + U_i^{J,n}\right) - \sum_{J\in\Lambda_I^B}\hat{T}_{IJ}^n U_i^{I,n} - \sum_{J\in\Lambda_I} C_j^{IJ,n}G_{ij}^{IJ,n} - \sum_{J\in\Lambda_I^B} D_j^{IJ,n}G_{ij}^{I,n} \\
&\quad - \sum_{J\in\Lambda_I} M_{ij}^{IJ,n}\left(E_j^{J,n} - E_j^{I,n}\right) - \sum_{J\in\Lambda_I}\alpha_{IJ}^n N_{ij}^{IJ,n}\left(U_j^{J,n} - U_j^{I,n}\right) = 0
\end{aligned} \tag{4.84}$$

is solved for every node I in the computational domain. The terms in this system are the time derivative (section 4.5.4), inviscid flux (section 4.3.2), inviscid flux on computational boundary (section 4.3.2), ALE flux (section 4.5.4), ALE flux on the computational boundary (section 4.5.4), viscous flux (section 4.4.2), viscous flux on the computational boundary (section 4.4.2), biharmonic artificial dissipation (section 4.3.3) and harmonic artificial dissipation (section 4.3.3) respectively. If the first order time discretization is used, the first, fourth and fifth term are replaced by the terms in equations (4.58), (4.75) and (4.76) respectively.

Chapter 5

Solution Procedures

5.1 Introduction

When the set of discrete equations has been produced, the unknowns have to be determined by a solution procedure. There are several possible candidates and a discussion of some competitive implicit and explicit solution schemes is presented. In this work, a nonlinear FAS multigrid scheme was selected and an extensive discussion of this method of multigrid convergence is given. The coarse meshes required for multigrid are constructed using an automatic procedure which is described in detail. The solver has been parallelized for use on computers with multiple processors and the strategy employed is explained in this chapter.

5.2 Review of Iterative Solution Procedures

The discrete systems of equations (4.28), (4.52) and (4.84), constructed in the previous chapter, can be written

$$R_i^I(\boldsymbol{U}) = S_i^I, \qquad I \in [1, N], \quad i \in [1, n] \tag{5.1}$$

where R_i^I is the residual of equation i at node I and S_i^I is a source term, independent of $\boldsymbol{U}$. For example, for the discretized steady–state Euler equations of equation (4.28),

$$\begin{aligned} V_I R_i^I \equiv \sum_{J \in \Lambda_I} \frac{C_j^{IJ}}{2} \left(F_{ij}^I + F_{ij}^J\right) + \sum_{J \in \Lambda_I^B} D_j^{IJ} F_{ij}^I \\ - \sum_{J \in \Lambda_I} m_{ij}^{IJ} \alpha_{IJ} \left(E_j^J - E_j^I\right) - \sum_{J \in \Lambda_I} \epsilon_2 \alpha_{IJ} N_{ij}^{IJ} \left(U_j^J - U_j^I\right) \end{aligned} \tag{5.2}$$

and

$$S_i^I \equiv 0. \tag{5.3}$$

Equation (5.1) is characterized by a dominant nonlinear convective term and a complicated interaction between the unknowns. The number of unknowns can be up to the order of tens of millions or higher and, in such large problems, the system

is usually quite stiff. The process of solving the discrete governing equations is, thus, a considerable challenge and can require excessive computer resources and solution times if not treated correctly. The different solution schemes in use at present are usually subdivided into two categories: *explicit* and *implicit*. In this context, implicit schemes are defined as solution procedures which require nontrivial matrix inversions to be performed, directly or iteratively. Explicit schemes on the other hand do not require the storage or inversion of matrices. Such schemes are constructed by localized vector products and summations, so that the updated solution is explicitly given as a function of known entities. For nontrivial problems, explicit solution procedures are iterative in nature, where many iterations are needed for convergence. While implicit solution procedures are capable of solving linear equations directly, the nonlinear discrete equations considered here also need some form of iteration process to converge. This is usually performed by applying implicit time stepping schemes of the form

$$\sum_{J=1,N} \left(\alpha I_{ij}^{IJ} + J_{ij}^{IJ,s} \right) \Delta U_j^{J,s} = S_i^I - R_i^{I,s}, \qquad I \in [1,N], \quad i \in [1,n] \tag{5.4}$$

where α is a relaxation parameter, I_{ij}^{IJ} is the identity matrix,

$$\Delta U_j^{J,s} = \left(U_j^{J,s+1} - U_j^{J,s} \right) \tag{5.5}$$

and $J_{ij}^{IJ,s}$ is the Jacobian at iteration level s, defined by

$$J_{ij}^{IJ,s} = \left. \frac{\partial R_i^I}{\partial U_j^J} \right|^s . \tag{5.6}$$

Implicit solution schemes, therefore, involve the inversion of a global Jacobian matrix. However, the number of iterations required for such iterative schemes is usually considerably less than that needed when explicit schemes are applied. It is, thus, not apparent whether the implicit or the explicit approach is the most efficient. One distinct advantage of explicit schemes is however the reduced storage requirements, since there are no matrices involved. Explicit schemes are also relatively easy to parallelize efficiently which is not always the case for their implicit counterparts.

5.2.1 Explicit Methods

A typical multistage explicit scheme for equation (5.1) can be written as

$$U_i^{I,s_p} = f_i^I(\boldsymbol{U}^{s_0}, \ldots, \boldsymbol{U}^{s_{p-1}}, \boldsymbol{U}^{(s-1)_0}, \ldots) \tag{5.7}$$

where the superscript s_p denotes the unknown of iteration s and subiteration p. The update of the unknown vector is thus an explicit function of known values. The probably most common explicit solution procedure is the P–stage Runge–Kutta scheme, which can be summarized as

$$U_i^{I,s_0} \quad = \quad U_i^{I,(s-1)_P}$$

$$
\begin{aligned}
U_i^{I,s_1} &= U_i^{I,s_0} + \alpha_1 CFL\Delta\tau_I \left(S_i^I - R_i^I(\boldsymbol{U}^{s_0})\right) \\
U_i^{I,s_2} &= U_i^{I,s_0} + \alpha_2 CFL\Delta\tau_I \left(S_i^I - R_i^I(\boldsymbol{U}^{s_1})\right) \\
&\cdot \\
&\cdot \\
&\cdot \\
U_i^{I,s_{(P-1)}} &= U_i^{I,s_0} + \alpha_{(P-1)} CFL\Delta\tau_I \left(S_i^I - R_i^I(\boldsymbol{U}^{s_{(P-2)}})\right) \\
U_i^{I,s_P} &= U_i^{I,s_0} + \alpha_P CFL\Delta\tau_I \left(S_i^I - R_i^I(\boldsymbol{U}^{s_{(P-1)}})\right)
\end{aligned}
\tag{5.8}
$$

The number of subiterations and the values of the Runge–Kutta constants, $\{\alpha_p\}$, can be optimized for time–accuracy, increased allowable CFL number or efficient smoothing of specified frequency ranges. As discussed in [90], the maximum allowable CFL number for stability is limited by $P-1$ for $P > 1$. The local time step $\Delta\tau_I$ is related to the discrete system and is independent of the time stepping scheme employed.

Explicit solution schemes usually require small memory overheads and are simple to implement. The domain of influence of a given node is however limited to a local region surrounding the node, dictated by the local stencil of the discretization scheme and the number of subiterations of the method. This means that several iterations are required for the correction of a given node to affect the remaining computational domain. These schemes are therefore efficient for problems where the solution influence domain is localized and the error frequencies are high. This is not the case for the fluid flow equations at high Reynolds-numbers, which are essentially hyperbolic outside the boundary layers.

5.2.2 Basic Implicit Methods

The linearization of equation (5.4) introduces linear systems of the form

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{5.9}$$

where $\boldsymbol{A} \in \mathbb{R}^{N\times n} \times \mathbb{R}^{N\times n}$ is assumed to be invertible. These systems are often too large to be solved by direct methods, such as Cholesky decomposition [30], even if they are sparse and bandwidth minimization algorithms are applied. The inversion is, thus, usually performed using an iterative approach. Some classic iterative methods are the Jacobi, Gauss–Seidel, Chebyshev and SOR methods [30]. Although these basic implicit schemes have been used in fluid dynamics computations, their convergence rate is usually considered to be inadequate for realistic calculations.

In many cases, a preconditioner can be applied to improve the performance of the basic solution scheme. A *left preconditioner* for the system of equations (5.9) consists of left–multiplying the equation with a matrix $\boldsymbol{P}$, so that

$$\boldsymbol{P}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{P}\boldsymbol{b} \tag{5.10}$$

is solved instead. The optimal left preconditioner would obviously be $\boldsymbol{A}^{-1}$, but by approximating the inverse, significant improvements in the behaviour of the

iterative scheme can be achieved. An alternative to left preconditioning is to apply a *right preconditioner*, $\boldsymbol{Q}$, of the form

$$(\boldsymbol{AQ})\left(\boldsymbol{Q}^{-1}\boldsymbol{x}\right) = \boldsymbol{b}. \tag{5.11}$$

This is essentially a variable transformation $\boldsymbol{x} \rightarrow \boldsymbol{Q}^{-1}\boldsymbol{x}$. A good preconditioner clusters the eigenvalues of the matrix close to unity, is easy to invert and only introduces a small memory increase in the solver. Some popular preconditioners are the ILU(n) [27, 83], SSOR [96] and LU-SGS [56] schemes.

5.2.3 Krylov Methods

Krylov methods are a class of popular implicit solution schemes for linear systems of the form of equation (5.9). The methods are based on a Gram–Schmidt orthogonal basis generation approach with the matrix inner product

$$< \boldsymbol{u}, \boldsymbol{v} >= \boldsymbol{u}^t \boldsymbol{A} \boldsymbol{v}. \tag{5.12}$$

Each iteration of the solver adds a new vector to the basis in which the solution of the linear system is allowed to exist. For an initial vector $\boldsymbol{v}_0$, the Krylov subspace of iteration s thus has the form

$$K_s = span\{\boldsymbol{v}_0, \boldsymbol{A}\boldsymbol{v}_0, \boldsymbol{A}^2\boldsymbol{v}_0, \ldots, \boldsymbol{A}^{s-1}\boldsymbol{v}_0\}. \tag{5.13}$$

The procedure is therefore guaranteed to converge in $N \times n$ iterations, but usually converges to within the desired tolerance much faster. The generic Krylov space solver can be summarized as follows:

1. A starting vector is chosen. This vector is the first basis vector in the Krylov subspace and is usually selected as $\boldsymbol{v}_0 = (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0)/|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0|$, the initial normed residual. The iteration number is set to $s = 0$.

2. A new guess of the unknown vector in K_s is constructed using linear combinations of the basis vectors in this space. The construction of the guess differs between the various Krylov methods.

3. The residual $\boldsymbol{r}^s = \boldsymbol{A}\boldsymbol{x}^s - \boldsymbol{b}$ of the current guess is checked whether it is within the desired tolerance. If it is, the iteration is considered converged, if not, a new dimension to the solution space is added by continuing to step 4.

4. The Krylov basis is expanded by adding the basis vector $\boldsymbol{A}^s\boldsymbol{v}_0$, and the iteration number is incremented, $s = s + 1$. The procedure returns to step 2.

Krylov space methods are constructed using matrix–vector products, vector inner products and vector–vector additions only. The schemes are thus readily parallelizable. For symmetric positive definite (SPD) systems, the *conjungate gradient* (CG) [30] method is usually applied. In fluid dynamics however, SPD systems usually only appear in incompressible flow problems, where the pressure increments are found by solving for a Poisson–type problem [106]. Due to

the nonsymmetric convective terms, schemes involving the Jacobian matrix are therefore not efficiently solved by the CG method. Alternative Krylov methods have consequently been developed to handle such problems. The most popular of these is probably the *generalized minimal residual* (GMRES) scheme [30, 84], applicable to general nonsymmetric indefinite systems. In this method, the solution increment is selected as the combination of basis vectors that minimizes the residual of the updated guess. The coefficients, $\{a_k\}$, of the increment vector

$$\boldsymbol{x}^s - \boldsymbol{x}^{s-1} = \boldsymbol{v}_k a_k, \qquad k \in [1, s] \tag{5.14}$$

are thus selected to minimize

$$R^s = ||\boldsymbol{A}\boldsymbol{x}^s - \boldsymbol{b}||^2 = \left|\left|\boldsymbol{A}\boldsymbol{v}_k a_k - \boldsymbol{r}^{s-1}\right|\right|^2. \tag{5.15}$$

By differentiating with respect to the coefficients, the equation

$$(\boldsymbol{A}\boldsymbol{v}_k)\,(\boldsymbol{A}\boldsymbol{v}_l)\,a_l = (\boldsymbol{A}\boldsymbol{v}_k)\,\boldsymbol{r}^{s-1} \tag{5.16}$$

results, which is solved to find the coefficients of the updated guess. This system has been shown to have a unique solution, as a result the GMRES procedure can not break down, and since the residual norm is minimized for each step, convergence is monotone [84].

The amount of work required and the memory usage of the GMRES method is strongly dependent on the number of search directions in use. It is therefore common to limit the number of search directions to the range $10 - 20$. When the number of maximum search directions is reached in the procedure, the search vectors are discarded and a new GMRES procedure is started with the latest guess for the solution vector as initial guess. This approach is termed the *restart* GMRES method.

An interesting feature of the GMRES method is the fact that the matrix only appears in the formulae as matrix–vector products. For fluid flow problems, the matrix is a combination of the Jacobian and a diagonal matrix. The matrix–vector product between the Jacobian and the increment vector can be approximated by

$$\sum_{J=1,N} J_{ij}^{IJ,s} \Delta U_j^{J,s} \approx \frac{R_i^I(\boldsymbol{U}^s + \epsilon\Delta\boldsymbol{U}^s) - R_i^I(\boldsymbol{U}^s)}{\epsilon} \tag{5.17}$$

where ϵ is a small value, thus avoiding the need to form any matrices. This results in a *matrix–free* GMRES scheme which can be considered explicit. The GMRES scheme is however dependent on efficient preconditioning for good performance. Since, in many cases, the preconditioning technique requires the storage of the Jacobian matrix, the memory usage required for the matrix–free GMRES solver may still be excessive. There are however recent developments of preconditioned GMRES methods, such as for example the LU–SGS–GMRES scheme described in [55], that do not require the storage of the non–diagonal terms in the Jacobian.

5.3 Multigrid Acceleration

The solution of a flow problem is usually characterized by the presence of a large number of different length scales. Placing an object in a moving fluid will introduce local changes in the flowfield close to the geometry as well as more global structures such as wakes and shocks dispersing throughout large regions of the flowfield. For complex geometries, the number of length scales in the problem will increase as the various components interact. Some regions of the flow are dominated by a local influence sphere, while other regions are critically dependent on the flow pattern some distance away. In the numerical system, this means that there exists a strong coupling between unknowns that are weakly coupled in the discretization. Explicit schemes will typically require a large number of iterations to transfer information between such points in the solution domain. Due to strong coupling, the amount of information that needs to be exchanged between these points is large. Thus, while explicit procedures are capable of quickly eliminating the local high error frequencies, the lower frequencies, reflecting errors of more global nature, are much slower to disappear in the solution iterations.

Multigrid methods [9, 21], were designed to effectively dampen the error of all frequencies simultaneously, while retaining the low operation count per iteration and low memory usage of the explicit schemes. There are two different kinds of multigrid approaches: the *geometric* (or *topological*) approach which works on the discretization level of the equations and the *algebraic* approach which considers the general linear system of equation (5.9). Geometric multigrid essentially consists of solving the discrete equations on several grids of different coarseness levels. Each mesh is responsible for removing a different interval of the error frequencies, with the coarse meshes damping the low frequency errors. As the complexity of the mesh decreases and the grid spacing is increased, the domain of influence of a given node will increase in size. If the coarsest meshes contain relatively few nodes, the domain of influence for any node will span most of the computational domain on each multigrid cycle. As a result, the coarse meshes are much more efficient in smoothing the low frequency errors than the fine mesh, since the low frequency errors of a fine mesh appear as high frequency errors on a coarse mesh [10]. An explicit relaxation scheme on a coarse mesh can thus effectively dampen out low error frequencies that slow down the convergence on the finest mesh. This is, in essence, due to the reduced system size and increased allowable time step on the coarse mesh. In addition, the computational cost per right–hand side evaluation is considerably lower on a coarse mesh than on a finer mesh.

Algebraic multigrid does not have a geometrical interpretation. Instead it works directly on a given matrix system. This invariably reduces the application of such schemes to linear problems, such as those arising from the implicit linearization of equation (5.4). Instead of solving discretized equations on coarser meshes, the algebraic multigrid approach generates several successively smaller matrix systems by summing rows in the matrices. The effect of this is essentially the same as for geometric multigrid, expanding the domain of influence of the unknowns and thus reducing the number of iterations required for explicit iterative solvers.

As mentioned above, there are two major routes that can be followed to solve the discrete system of equations of fluid flow problems. One is to linearize the problem, using equations such as (5.4), leading to implicit schemes which introduce the Jacobian matrix into the calculations. Alternatively, the nonlinear equation (5.1) can be solved directly. Multigrid can be used in conjunction with either of these approaches as it is efficient in solving both linear as well as nonlinear equation systems. It is found [31] that the asymptotic convergence behaviour of linearized and nonlinear multigrid is the same. However, a numerical comparison of multigrid performance of these two approaches on fluid flow equations has been conducted [61], concluding that the linearization procedure is somewhat more efficient. On the other hand, the need for storing the Jacobian increases the memory usage of the solver significantly. It was therefore decided to implement the nonlinear solver for the work presented in this report.

For geometric multigrid, there are two main approaches in use: the *correction multigrid* (CMG) scheme and the *full approximation storage* (FAS) scheme. The CMG scheme is only applicable to linear problems, but is illustrative of the multigrid methodology and is briefly described in the following. Starting from a linear discrete problem

$$\boldsymbol{L}_f(\boldsymbol{U}_f) = \boldsymbol{S}_f \tag{5.18}$$

an initial guess of cycle number s on the fine mesh, $\boldsymbol{U}_f^s$, is given and a relaxation step by a method of choice is performed, creating an updated guess $\boldsymbol{U}_f^{s^*}$. In principle, any relaxation scheme may be used, but for good multigrid performance, it is essential that the scheme efficiently dampens out high–frequency errors. One can therefore write

$$\boldsymbol{U}_f = \boldsymbol{U}_f^{s^*} + \boldsymbol{E}_f^s, \tag{5.19}$$

where the error $\boldsymbol{E}_f^s$ can be assumed to consist only of low frequency contributions. The *residual* or *defect* at the initial multigrid cycle is given as

$$\boldsymbol{D}_f^{s*} = \boldsymbol{L}_f(\boldsymbol{U}_f^{s^*}) - \boldsymbol{S}_f. \tag{5.20}$$

By subtracting this equation from equation (5.18), and applying the assumption of linearity, i.e. using

$$\boldsymbol{L}_f(\boldsymbol{U}_f) - \boldsymbol{L}_f(\boldsymbol{U}_f^{s^*}) = \boldsymbol{L}_f(\boldsymbol{U}_f - \boldsymbol{U}_f^{s^*}), \tag{5.21}$$

one obtains the equation

$$\boldsymbol{L}_f(\boldsymbol{E}_f^s) = -\boldsymbol{D}_f^{s^*} \tag{5.22}$$

for the error. Since it is assumed that the error is smooth on the fine grid, it can be concluded that the equation can be accurately solved on a coarser discretization. The discrete system on a coarse mesh consists of fewer unknowns than that of the fine mesh, typically 1/4 in two dimensions and 1/8 in three dimensions, thus reducing the computational effort required compared to fine mesh iterations. Also, an explicit relaxation scheme will be more efficient on the coarse mesh, since

the stability restrictions are reduced and the domain of influence is increased, facilitating the efficient solution of lower frequencies. Solving for the error on the coarse mesh is therefore much faster than if the fine mesh discretization is retained. By introducing a coarse mesh representation of the error, $\boldsymbol{E}_c^s$, the equation

$$\boldsymbol{L}_c(\boldsymbol{E}_c^s) = -\boldsymbol{I}_c^f \boldsymbol{D}_f^{s^*} \tag{5.23}$$

is solved on the coarse mesh. Here, the coarse mesh level operator $\boldsymbol{L}_c$ has been introduced. This is a discretization of the governing equations on the coarse mesh level. Also, $\boldsymbol{I}_c^f$ is a mapping of variables from the fine mesh to the coarse mesh, termed the *restriction operator*. A band of frequencies considered low on the fine mesh are represented as high frequencies on this coarse mesh level. These frequencies can now be efficiently dampened by a relaxation step on the coarse mesh. If there are even lower frequencies present in the solution field, i.e. if the coarse mesh level can be further coarsened, a multigrid method can be applied on equation (5.23) itself and so on until all frequency ranges can be efficiently smoothed by respective mesh levels. Solving equation (5.23) thus results in a correction on the coarse mesh level $\boldsymbol{E}_c^{s^*}$. This approximation of the low error frequencies is then mapped up to the fine mesh level and added to the fine mesh solution according to

$$\boldsymbol{U}_f^{s+1} = \boldsymbol{U}_f^{s^*} + \boldsymbol{I}_f^c \boldsymbol{E}_c^{s^*} \tag{5.24}$$

where $\boldsymbol{I}_f^c$ is a coarse to fine mapping, termed the *prolongation operator*. The CMG scheme thus directly operates on the error of the linear equation. An important point to note is that if the residual of the finest mesh equation vanishes, i.e. the solution is found, the multigrid scheme will not add any correction to the unknowns. This consistency property of multigrid schemes ensures that if a unique solution exists of the numerical systems on all mesh levels and if the solution increment is zero, the method has converged to the right solution.

5.3.1 FAS Multigrid

For nonlinear equation systems, equation (5.21) is invalid. For such problems, a different multigrid strategy must be applied. The FAS multigrid scheme [9] is capable of handling nonlinearity and is used in the work presented in this report. The nonlinear discrete system

$$\boldsymbol{R}_f(\boldsymbol{U}_f) = \boldsymbol{S}_f \tag{5.25}$$

on the fine mesh is considered, with the solution approximated at cycle s by $\boldsymbol{U}_f^s$. A relaxation step is performed, damping high error frequencies, producing the improved guess $\boldsymbol{U}_f^{s^*}$. The objective is now to find the correction $\boldsymbol{E}_f^s$ such that

$$\boldsymbol{U}_f = \boldsymbol{U}_f^{s^*} + \boldsymbol{E}_f^s. \tag{5.26}$$

The fine grid error can be assumed to consist mainly of low frequencies since the fine mesh relaxation is usually very efficient at eliminating the highest frequencies

on the fine mesh. One can thus accurately approximate the fine–grid error on the coarse mesh. Again, the defection of the fine mesh for the improved guess on multigrid cycle s is defined as

$$\boldsymbol{D}_f^{s^*} = \boldsymbol{R}_f(\boldsymbol{U}_f^{s^*}) - \boldsymbol{S}_f. \tag{5.27}$$

By subtracting this equation from (5.25) and using (5.26), the generalization

$$\boldsymbol{R}_f(\boldsymbol{U}_f^{s^*} + \boldsymbol{E}_f^s) - \boldsymbol{R}_f(\boldsymbol{U}_f^{s^*}) = -\boldsymbol{D}_f^{s^*} \tag{5.28}$$

of equation (5.22) valid for nonlinear operators is found. Since the error can be considered smooth, it is more efficient to solve this equation on a coarse mesh. Instead of solving for the equation error on the coarse meshes, as in the CMG scheme, however, a coarse grid variable is introduced, defined as

$$\boldsymbol{U}_c^s = \boldsymbol{I}_c^f \boldsymbol{U}_f^{s^*} + \boldsymbol{E}_c^s \tag{5.29}$$

where $\boldsymbol{E}_c^s$ is the coarse mesh representative of the low frequency error components of the iteration. The coarse mesh analogy of equation (5.28) is thus given by

$$\boldsymbol{R}_c(\boldsymbol{U}_c^s) = \boldsymbol{R}_c(\boldsymbol{I}_c^f \boldsymbol{U}_f^{s^*}) - \boldsymbol{I}_c^f \boldsymbol{D}_f^{s^*}. \tag{5.30}$$

This equation serves the same purpose as equation (5.23) in the CMG scheme. However, equation (5.30) is solved for a coarse mesh unknown variable instead of the coarse mesh interpretation of the fine mesh error. By defining

$$\hat{\boldsymbol{S}}_c = \boldsymbol{R}_c(\boldsymbol{I}_c^f \boldsymbol{U}_f^{s^*}) - \boldsymbol{I}_c^f \boldsymbol{D}_f^{s^*}, \tag{5.31}$$

the coarse mesh equation can be written,

$$\boldsymbol{R}_c(\boldsymbol{U}_c^s) = \hat{\boldsymbol{S}}_c, \tag{5.32}$$

which is of the same form as equation (5.25). In essence, this means that a very similar equation can be solved on the coarse meshes as on the fine mesh, with the only difference being a modified source term. This is a great advantage implementationally, since the majority of the numerics are related to the evaluation of $\boldsymbol{R}$, making it possible to use many of the same subroutines on all mesh levels.

The coarse mesh equation (5.32) can now be relaxed, yielding an improved guess $\boldsymbol{U}_c^{s^*}$ of the coarse mesh unknowns. Since it is of the same form as equation (5.25), exactly the same steps can be performed to solve the coarse mesh equation by the FAS multigrid method. The procedure thus naturally introduces as many mesh levels as is required to resolve all frequencies in the solution. An approximation of the coarse mesh error is now given by

$$\boldsymbol{E}_c^{s^*} = \boldsymbol{U}_c^{s^*} - \boldsymbol{I}_c^f \boldsymbol{U}_f^{s^*}. \tag{5.33}$$

The fine mesh correction formula thus has the form,

$$\boldsymbol{U}_f^{s+1} = \boldsymbol{U}_f^{s^*} + \boldsymbol{I}_f^c \boldsymbol{E}_c^{s^*}, \tag{5.34}$$

which is completely analogous to equation (5.24). Again, it is easily seen that the FAS scheme is consistent.

5.3.2 Multigrid Performance

Multigrid has been shown to give excellent performance for the solution of elliptical problems, such as the Poisson equation, with convergence rates of about 0.1 [10], i.e. one order of residual reduction per multigrid cycle. This convergence rate has been shown to be independent of mesh size, which is quite a remarkable feature, meaning that the computational effort of solving a system with N unknowns is proportional to N. The multigrid procedure is therefore said to be $\mathcal{O}(N)$ for this problem. For the Poisson equation, multigrid therefore has the same operation count as inverting a diagonal matrix, up to a constant independent of N, and can be considered of optimal order.

The optimal convergence rate achievable for the multigrid method on the hyperbolic fluid flow equations has been shown to be 0.75 for second order schemes [70]. Even this reduced convergence rate is however seldom observed. The mesh independency found in elliptic problems is, however, still satisfied if the mesh quality is sufficiently good. An example of this is illustrated in Figure 5.1 where the same testcase is run on two meshes of different size. The small mesh consists of $4,298$ nodes while the large mesh consists of $10,673$ nodes. The large and small meshes employed six and five mesh levels respectively for the multigrid procedure. It can be seen that multigrid convergence is largely mesh–independent in this case, both for the subsonic and the transonic calculations. This can be compared with Figure 5.2 which shows the $\mathcal{O}(N^2)$ nature of an explicit Runge–Kutta time stepping scheme on the subsonic problem, using the same two meshes. It can, however, be observed in Figure 5.1 that, while the number of cycles required for convergence for a given problem is essentially mesh–independent, the convergence rate is dependent on the flowfield characteristics. Multigrid usually performs best on smooth problems and the convergence rate deteriorates when high gradients appear in the solution. This problem can sometimes be alleviated by applying more advanced relaxation operators [63, 78], but this is not discussed further here.

Another important factor in the convergence behaviour of multigrid schemes is the mesh quality. The introduction of highly stretched meshes in viscous high Reynolds–number calculations is known to deteriorate the convergence rate of the multigrid approach [10, 63]. This is due to the fact that the unknowns are much more strongly connected in the direction of small grid spacing, i.e. the direction normal to the wall, than in the other directions where the grid spacing can be several orders of magnitude larger. A standard explicit relaxation scheme is only capable of efficiently damping out errors in the direction of strong connectivity, since the local stability limit is dictated by the mesh spacing in this direction. This essentially means that the multigrid mesh independency is lost for such problems unless special care is taken, such as the introduction of more complex relaxation procedures like line–relaxation or Krylov space methods. Another approach to improve multigrid performance on stretched meshes, is to modify the coarse mesh generation procedure. As reported in [63, 70], the coarse meshes can be constructed so as to improve multigrid stability and convergence rates for highly stretched meshes. The method in [70] consist of creating a hierarchy of structured meshes which employ semi–coarsening in all directions simultaneously.

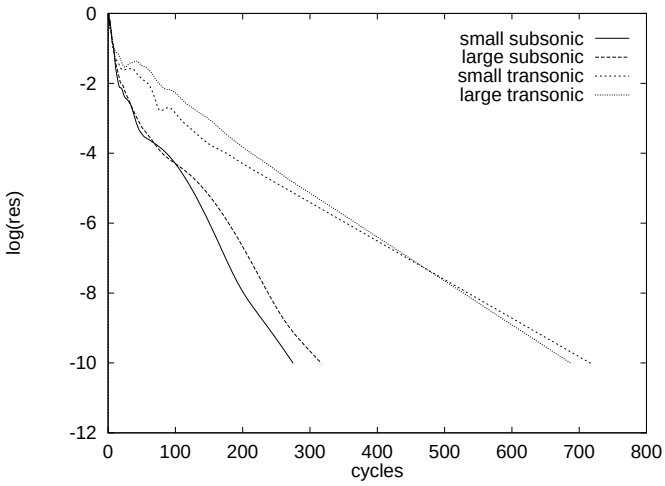

Figure 5.1: Comparison of convergence curves for a multigrid procedure on two different meshes for subsonic and transonic inviscid flow around a NACA0012 aerofoil.

Although $\mathcal{O}(N)$, this scheme requires the generation of a large number of coarse meshes, especially in three dimensions. The unstructured mesh framework makes it possible to coarsen in directions of the small grid spacing only (i.e. in directions with large edge coefficients) in regions of mesh anisotropy. This approach has been shown to improve convergence rates, but increases the complexity of the coarse meshes. If coarsening is only performed in the direction normal to the walls on the initial coarse mesh levels, the reduction in nodes for the coarse mesh in these regions will only be of a factor of 2, compared to 8 for three–dimensional homogenous coarsening. In many cases the increased multigrid efficiency resulting from directional coarsening improves the overall convergence performance however, even though the complexity of the multigrid cycles is increased.

For inviscid schemes, the discrete operator on the coarse mesh levels can be simplified to reduce the operation count for a multigrid cycle. This can be achieved by using normal first–order accurate Laplacian smoothing instead of the costly fourth–order dissipation term on the coarse meshes. As long as the scheme converges, there is of course no implication on solution accuracy with this approach. It was, however, found that this approach is unstable if applied to viscous flows involving highly stretched meshes. In such cases, the discretization schemes applied on the fine and coarse meshes are essentially the same.

5.3.3 Memory Usage

Single grid solvers store the bulk of the data needed in two types of arrays; edge arrays and node arrays. Since the number of edges in a mesh is approximately proportional to the number of nodes, one can write the memory usage of a scheme on a mesh G as a linear function of the number of nodes, $mem(G) = KN_n$ where N_n is the number of nodes in the mesh. For the three–dimensional turbulent solver, the memory usage in bytes associated with the finest grid is approximately

$$mem^1(G) \approx 620N \tag{5.35}$$

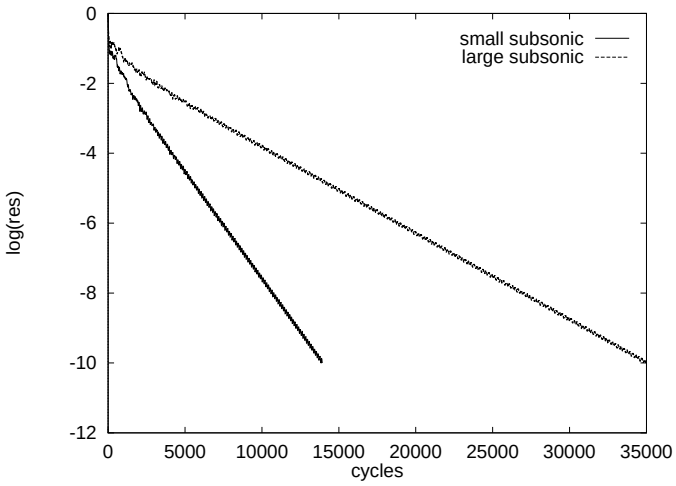

Figure 5.2: Illustration of the mesh dependency of an explicit, single grid solution procedure applied to subsonic inviscid flow around a NACA0012 aerofoil.

for a simplex mesh and

$$mem^1(G) \approx 560N \tag{5.36}$$

for a hybrid mesh if double precision floating–point arrays are used. By assuming a worst–case scenario coarsening ratio of 2 and using the fact that inter–grid operators require the storage of about 12 bytes per fine node, the memory usage for a scheme applying M meshes is estimated as

$$mem^M(G) \approx 2(mem^1(G) + 12)\left[1 - \left(\frac{1}{2}\right)^M\right] \tag{5.37}$$

where N_n denotes the number of fine grid nodes. This means that the memory usage for a multigrid solver is usually less than twice that of a single grid solver. However, since significant portions of the allocated memory of single–grid schemes are used to store intermediate data, the memory usage of a multigrid scheme can be further reduced by re–using these help arrays on the coarser mesh levels. In such a way, the extra amount of memory required for a multigrid scheme can usually be reduced to about 40% of that of a single grid solver, i.e.

$$mem^M(G) \approx 1.4mem^1(G). \tag{5.38}$$

This is a very attractive feature of the multigrid scheme compared to many other advanced solvers that need to store matrices. If, for example, an iterative approach is used which requires a Jacobian matrix to be stored, an additional memory usage of about 200 bytes times the number of connectivities in the mesh is needed if the turbulence model is solved separately. Typically, the memory usage for such a solver is at least four times that of the explicit scheme.

5.3.4 Cycling Schemes

If more than two grids are employed in the multigrid procedure, there are several alternative ways of traversing the coarse meshes. The selection of the cycling scheme will influence the intensity of the relaxation on the different coarseness levels, and thus control the relative damping of the different error frequencies. It is also possible to control the way a scheme damps out frequencies by varying the number of relaxation steps between the meshes. In this work, only one three–stage Runge–Kutta step is performed on each relaxation however. Also, relaxation is only performed in the coarsening direction of the cycling scheme, Figure 5.3. The two main cycling schemes in use at present are the V and W cycles, illustrated in Figure 5.3. The number of smoothing steps for grid level m is

$$\sigma^m = 1 \tag{5.39}$$

for the V cycle and

$$\sigma^m = \begin{cases} 1, & m = 1 \\ 2\min(m-1, M-2), & m > 1 \end{cases} \tag{5.40}$$

for the W cycle. The W cycle is thus clearly more complex than the V cycle and requires considerably more computational effort. On the other hand, relaxation on the the coarse mesh levels is stronger, so normally fewer multigrid cycles are needed for convergence. The selection of the cycling scheme is thus best performed on the criteria of CPU time usage for convergence. A comparison was performed between different multigrid schemes for a simple two–dimensional inviscid subsonic problem of flow around an airfoil. The Mach number is 0.63 and the angle of attack is one degree. The mesh consists of 3933 nodes. In Figure 5.4, a plot of the density residual versus CPU time for the different cycling schemes is plotted. It is seen that, in this case, the *5-g*V cycle, also termed the $5-F$ cycle in the literature, Figure 5.3, yields the best performance. For the gV scheme, the number of relaxations for grid level m is given by the formula

$$\sigma^m = \min(m, M-1) \tag{5.41}$$

and is therefore located between the V and W cycles in complexity. This scheme has shown itself to be a good compromise between CPU–cost and efficient coarse–mesh error smoothing and is therefore used exclusively in the examples shown in this report.

It is possible to vary the number of grids in use in the cycling schemes in the solution process. The most common way of doing this is by the *full multigrid* (FMG) scheme, where the multigrid solution is first converged on one of the coarser meshes, and then expanded to successively include finer mesh levels. In this way, a cheap solution can be found on a coarse mesh level and this can be used as an initial guess on the fine mesh. If adaptive refinement is used in the solution procedure, the refined meshes can be treated as successive fine meshes for a FMG algorithm. This approach was not investigated in the present work however.

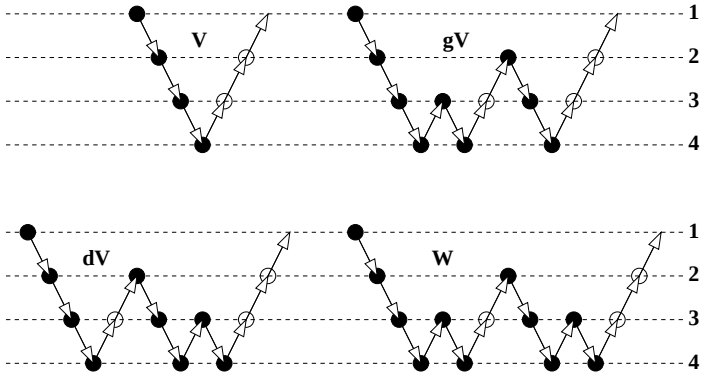

Figure 5.3: Illustration of different cycling scheme stencils for the multigrid procedure, shown for the four mesh case. The filled circles indicate grid visits where a relaxation step is performed. The variables are not relaxed at the positions indicated by the open circles. Grid number 1 denotes the finest mesh level and grid number 4 is the coarsest.

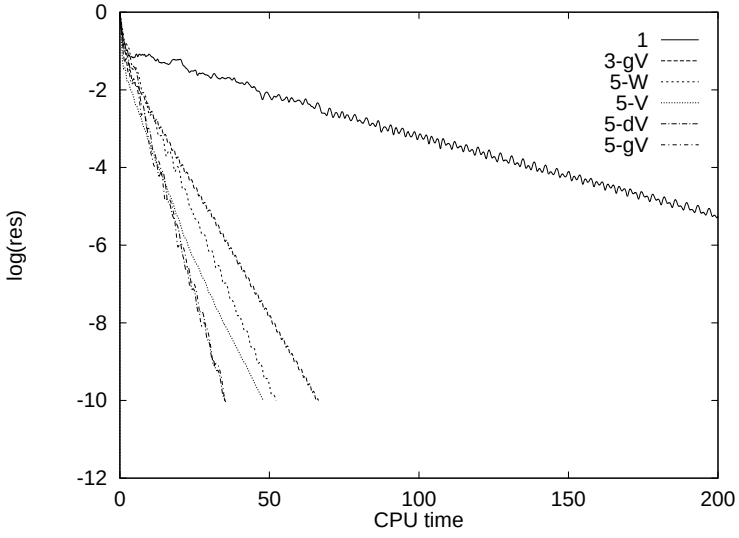

Figure 5.4: Convergence curves for inviscid subsonic flow around a NACA0012 aerofoil.

5.3.5 Relaxation Schemes

The relaxation scheme used in this report is a three–stage Runge–Kutta approach with local time stepping (section 5.2.1). The values

$$\alpha_1 = 0.6, \quad \alpha_2 = 0.6, \quad \alpha_3 = 1.0 \tag{5.42}$$

are used for the Runge–Kutta coefficients. To reduce the computational requirements of the scheme, the viscous and artificial dissipation terms are only calculated once every relaxation step, using the initial unknowns of the iteration U_i^{I,s_0}. The stability limit for this scheme is $CFL = 1.8$.

For the coarse mesh equations, the initial guess

$$\boldsymbol{U}_c^s = \boldsymbol{I}_c^f \boldsymbol{U}_f - \Delta\boldsymbol{\tau} \boldsymbol{I}_c^f \boldsymbol{D}_f^{s^*} \tag{5.43}$$

is used where $\Delta\boldsymbol{\tau}$ is a diagonal matrix with the local timesteps on the diagonal. This yields $20-30\%$ faster convergence compared to simply using the interpolated unknowns. Since only the fine mesh defect is used, the consistency of the scheme is not affected.

The local time step values are determined according to

$$\Delta\tau_I = V_I \left[\sum_{J \in \Lambda_I} \left(\left|\left| C_j^{IJ} \right|\right| \lambda_{max}^I + \frac{2 \left|\left| C_j^{IJ} \right|\right|^2 \left(\nu_I + \nu_I^t\right)}{V_I} \right) \right]^{-1} \tag{5.44}$$

where

$$\lambda_{max}^I = \frac{\left| u_j^I C_j^{IJ} \right|}{\left|\left| C_j^{IJ} \right|\right|} + c_I. \tag{5.45}$$

The two terms in the brackets are the inviscid and viscous contributions respectively. The inviscid part was originally devised in [28] from energy stability criteria. The viscous part is added as in [**?**] with the addition of turbulence effects through the turbulent viscosity at node I, ν_I^t [59]. For time–accurate calculations, a correction

$$\Delta\tau_I^* = \frac{2\Delta\tau_I}{2 + \beta\frac{\Delta\tau_I}{\Delta t}} \tag{5.46}$$

is included, where β is the coefficient that multiplies the implicit variable for the given scheme, i.e. $\beta = 1$ for the first order scheme of equation (4.58) and $\beta = 3/2$ for the second order scheme of equation (4.59). This correction is motivated by considering stability analysis of the equation resulting from the limit where the time term becomes dominant.

For the Spalart–Allmaras turbulence model, a different local time step is applied. This time step was deduced from the same principles as that of the governing equations, so that

$$\Delta\tau_I^t = \frac{2\Delta\hat{\tau}_I^t}{2 + S_I^t \Delta\hat{\tau}_I^t} \tag{5.47}$$

where

$$S_I^t = \left|\partial_k^h \tilde{u}_k^I\right| + c_{b1}\left(\tilde{\omega}_I + (1+c_{t3})\frac{\tilde{\nu}_I}{\kappa^2 d^2}\right) + c_{w1}\frac{\tilde{\nu}_I}{d^2} \tag{5.48}$$

is the source term correction and

$$\Delta\hat{\tau}_I^t = V_I\left[\sum_{J\in\Lambda_I}\left(\left|\left|C_j^{IJ}\right|\right|\lambda_{max}^{I,t} + \frac{2\left|\left|C_j^{IJ}\right|\right|^2\left(\nu_I + 3\nu_I^t\right)}{V_I}\right)\right]^{-1} \tag{5.49}$$

are the contributions from the convective and diffusive terms. Here

$$\lambda_{max}^{I,t} = \frac{\left|u_j^I C_j^{IJ}\right|}{\left|\left|C_j^{IJ}\right|\right|} + \eta, \tag{5.50}$$

where the parameter η is added to ensure stability in boundary layers, where the velocities are small. This variable was set to

$$\eta = 0.1u_\infty \tag{5.51}$$

in all calculations.

5.3.6 Intergrid Mappings

The intergrid mappings are important components of any geometric multigrid scheme and the selection of these mappings has direct consequences on the effectiveness and stability of the multigrid implementation. An important characteristic of the intergrid mappings are their order. As described in [31], the order of a prolongation operator is said to be m_p if the operator interpolates polynomials of degree $m_p - 1$ exactly. The order of a restriction operator, m_r is defined as the order of the adjoint prolongation operator. For a differential operator of order $2m$, it is stated in [31] that the inequality

$$m_p + m_r > 2m \tag{5.52}$$

should be satisfied for good multigrid behaviour.

In the current work, a conservative volume–weighted restriction operator

$$U_I^c = \frac{1}{V_I^c}\sum_{J\in M_I} V_J^f U_J^f \tag{5.53}$$

is used, where M_I is defined as the set of fine–grid nodes contained within the coarse mesh control volume. This formula builds on the assumption that the coarse mesh level control volume is completely defined by the collection of fine–grid control volumes associated with the nodes in M_I, i.e. if the grid levels are *nested*. This is the case for the scheme presented here, since the coarse meshes are generated by an agglomeration procedure. For prolongation, the injection scheme

$$U_I^f = U_J^c, \quad \forall I \in M_J \tag{5.54}$$

is used. The coarse mesh value is thus simply inserted into the fine mesh nodes associated with the coarse mesh control volume. These mapping strategies are the same as those used in [90], where it is shown that they are adjoint.

While the inequality of equation (5.52) is satisfied for the Euler equations, it appears to be violated if the viscous terms are included. In search of better convergence rates, preliminary studies employing linear prolongation were also performed. No significant improvement in convergence rate was found, but the multigrid scheme was found to be less stable with this approach. The use of higher order intergrid mappings was therefore not investigated further.

5.3.7 Turbulence Speedup

The Spalart–Allmaras turbulence model has shown itself to be slow to converge in many cases. This is a result of the way in which the turbulent viscosity field evolves with the relaxation iterations. The only term hindering a homogenous solution of the model is the trigger term, initiating turbulence at user–specified locations of the flow. The domain of influence of this term is however quite small, essentially perturbing the flow in the immediate region of the trigger point or line and counting on the convective and source terms of the model to translate and strengthen the turbulent viscosity downstream. The trigger term is also often relatively slow to build up which leads to an initial convergence towards a laminar solution, before the turbulent effects start evolving. It was however found that this adverse quality can easily be countered by modifying the initial conditions of the problem. Instead of initiating the calculations with a uniform field of small turbulent viscosity, the turbulent viscosity can be set to a prescribed value, typically 25 times the laminar viscosity, in a specified region around the trigger lines for the turbulence model. This approach can be thought of as an *active* triggering procedure. In this way, the turbulence field is allowed to evolve immediately, speeding up convergence considerably, as illustrated in Figure 5.5. If uniqueness is assumed, this will not affect the final solution.

5.4 Coarse Mesh Generation

Traditionally, the coarse meshes for the multigrid acceleration procedure on unstructured meshes have been generated manually by applying unstructured mesh generators at each mesh level [40, 62, 76]. This approach is, however, quite time–consuming for the user, since a certain coarseness ratio, typically doubling the spacing in each direction for homogenous meshes, has to be achieved if multigrid performance is to be optimal. This not only takes time, but also introduces mesh generation problems for the coarse mesh levels. This is especially true for complicated geometries where the surface geometry will be difficult to capture and recovery may fail if a Delaunay procedure is used or closing the domain will be difficult if advancing front schemes are employed. Usually this limits the number of meshes that can practically be applied, which again results in poor smoothing behaviour of low frequencies for the multigrid method. Typically no more than three meshes are used when the manual coarse mesh generation strategy is applied. In addition, these meshes will in general not be nested, which complicates

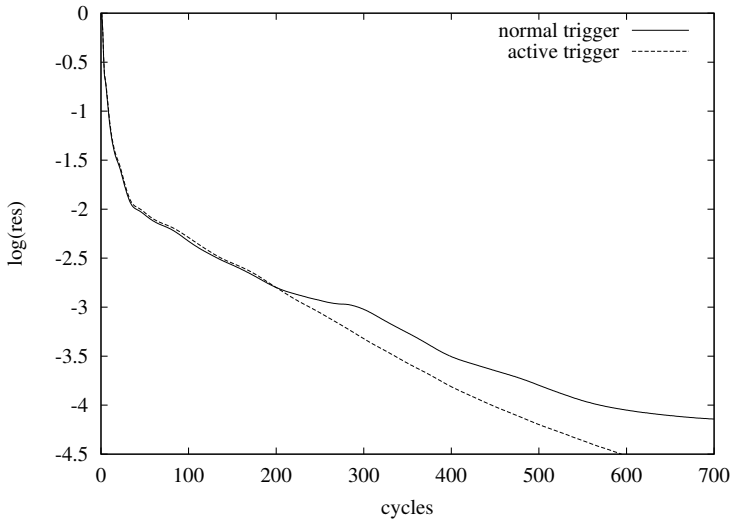

Figure 5.5: Comparison of convergence curves for turbulent flow around a wing with and without active triggering of the turbulent field.

inter–grid mappings.

5.4.1 Automatic Procedures

The problems associated with manual coarse mesh creation have inspired research into automatic coarsening and refinement methods, where minimal user input is needed in the generation of the multigrid mesh cascade. One approach [5, 12, 77] is to start with a very coarse mesh and successively refine until an appropriate mesh spacing is achieved. A disadvantage of this approach is that the creation of the coarsest mesh may be a challenge for any unstructured mesh generator on complex geometries. More importantly, the finest mesh is, in a sense, dictated by the initial coarse mesh. Special care thus needs to be taken on the coarse mesh to achieve the desired mesh quality on the finest mesh which is paramount to the solution accuracy.

An alternative approach is to first generate the finest mesh and then successively coarsen this mesh until the number of meshes desired is obtained. Following this approach, the solution accuracy is under the direct control of the user, through the generation of the initial mesh. There are several alternatives of such strategies, the two most common are Voronoi coarsening and agglomeration. The Voronoi coarsening method consists of collapsing edges in the mesh, locally regenerating the coarse mesh after each fine mesh edge removal [15, 67]. Since tetrahedral elements are created on all mesh levels with these methods, there can be problems associated with retaining good geometry descriptions on the coarser levels and the introduction of hybrid meshes is nontrivial for most of these methods. Instead of creating coarse meshes in the traditional sense, the grid agglomeration approach employed here works on the dual mesh by merging control volumes. The introduction of hybrid meshes is, therefore, irrelevant for agglomeration schemes with respect to robustness. Indeed, the scheme is guar-

anteed to generate a valid coarse dual, as long as the initial dual mesh is valid. There are, however, issues related to coarse mesh quality which are dependent on the original mesh structure. The absence of node definitions on the coarse meshes and the rather diffuse geometric interpretation of coarse mesh control volumes can also have a negative influence on the practical implementation of the solution procedure on coarse agglomerated meshes.

5.4.2 Grid Agglomeration

The agglomoration approach works on the dual mesh, merging control volumes in a way to achieve the local coarseness ratio required [57, 63]. This approach is purely edge–based and can therefore be applied on any mesh type after the edge–based data structure has been assembled. Since no mesh generation issues are raised in the procedure, the scheme is completely stable. It is also fast, typically requiring the CPU time equivalent to 1–3 multigrid cycles, requires little memory overhead and produces nested meshes which simplify the inter–grid mappings. As mentioned earlier, the introduction of hybrid meshes is trivial from an implementation point of view since the method is edge–based. Grid agglomeration does not generate coarse meshes in the classical sense, but rather generates a series of edge–based data structures that can be used by the multigrid solver. This has been shown to be completely analogous to algebraic multigrid methods for linear problems, where in essence the coarse mesh equations are generated by adding equations in the fine mesh problem [60]. For the nonlinear discrete systems considered in this report there is no direct relationship, but this illustrates the algebraic nature of the agglomeration procedure. The problems of coarse grid validity and accurate geometry representation often arising in geometric coarse mesh generation approaches are therefore eliminated using this approach. The homogenous agglomeration scheme can be summarized as follows:

1. A seed node is selected from a seed list, denoted by the open circle in Figure 5.6(a).

2. The nodes connected to the seed node by edges that haven't already been merged are grouped together creating a supernode, Figure 5.6(b).

3. When the above procedure is completed for all nodes, internal edges in the supernodes are deleted and edges bordering the same two supernodes are merged into superedges, Figure 5.6(c). This is done by adding the coefficients defined in equation (4.6) of the merged edges. In a similar way, the boundary coefficients defined by equation (4.7) are added.

When the agglomeration loop is complete, supernodes only connected to one edge are identified and merged using this connectivity. This can occur if a supernode control volume is completely surrounded by another control volume and, possibly, a part of the boundary of the computational domain.

There are different choices available for the priority list of nodes used in the agglomeration process. The simplest approach is to select seed nodes randomly in the mesh. This approach usually works quite well for simplex meshes, where the connectivities of each node is high. For such meshes, the desired coarseness

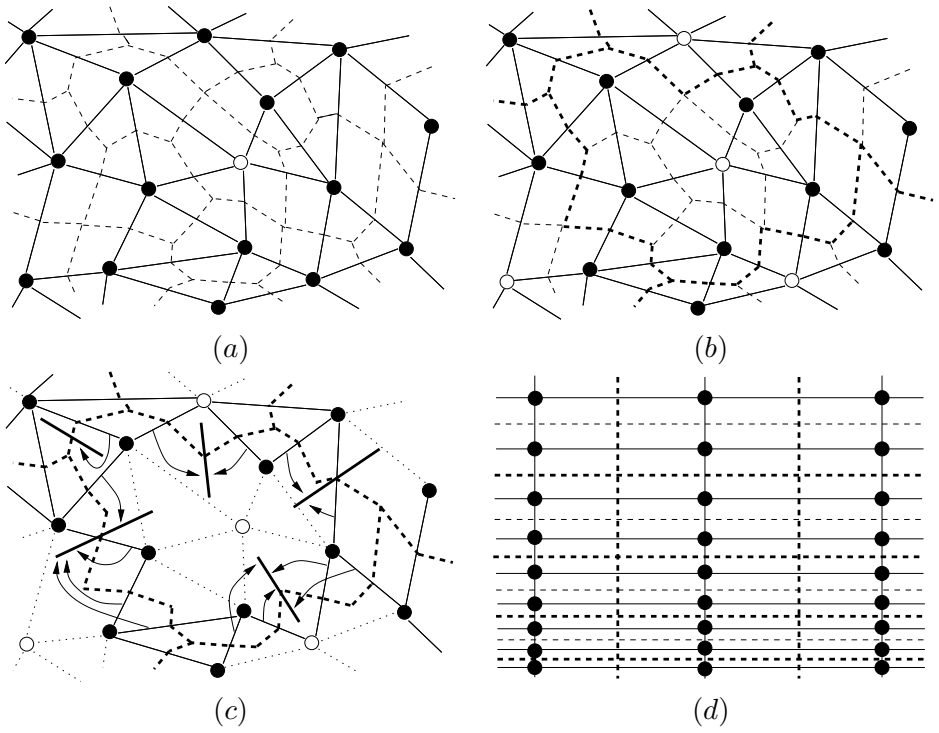

Figure 5.6: Illustration of the agglomeration procedure. The mesh is shown in solid lines. The dual mesh is denoted by dashed lines for the fine mesh and bold dashed lines for the coarse mesh.

ratio is closely obtained with random seed node selection. For hybrid meshes however, the complexity of the coarse meshes may be considerably higher than desired due to the reduced connectivity. A better approach, for such meshes, is to select seed nodes according to a connectivity criterion by creating a seed priority list in which the nodes of largest connectivities are selected first. The list is updated as the agglomeration takes place, so that edges connected to nodes that have already been agglomerated are not taken into account in the ordering. This approach usually results in somewhat larger coarseness ratios. There is also a more elaborate scheme available, tailored for hybrid meshes, [32]. This approach is constructed to generate as regular coarse meshes as possible by moving beyond the nearest neighbours in the agglomeration procedure. The effect such schemes have on multigrid performance is still under investigation and the approach was not attempted in the current implementation. Usually, it is advisable to use the boundary nodes as seeds, since this improves the coarse mesh structure close to the boundaries. The boundary nodes are, therefore, treated separately in the agglomeration process and the selection of internal node seeds does not start until all boundary nodes are processed. The agglomeration of the boundary nodes is performed as for the internal nodes, i.e. randomly or by connectivity criteria.

A series of dual meshes generated for an inviscid aerofoil mesh is shown in Figure 5.7. It has to be noted however that the dual meshes on the coarse grid

levels are not uniquely defined. Strictly speaking only the edge coefficients and control volume sizes are known on the coarse meshes. The dual meshes plotted in this figure are therefore just the remnants of the original dual mesh where the control volume boundaries have been retained. A discussion of the appearance of the coarse mesh discretization resulting from the multigrid procedure is given in [60].

In some cases, such as for higher–order inter–grid mappings and the compact viscous stencil, there is a need for nodal coordinates on the coarse meshes. This information is not well defined in the agglomeration procedure, but it was found that mapping the coordinates of the finest grid level using equation (5.53) to the coarser levels produces well–behaved schemes.

5.4.3 Directional Agglomeration

Multigrid performance can be seriously degraded when applied on highly anisotropic meshes. As discussed in section 5.3.2, the convergence rate on a given mesh can be improved if a directional mesh coarsening scheme is applied. This can easily be implemented by modifying a homogenous agglomeration code. In essence, only step 2 in the agglomeration procedure described above needs to be replaced as follows

1. ...
2. The nodes connected to the seed node by edges that haven't already been merged and satisfy the directionality condition are grouped together creating a supernode.
3. ...

The directionality condition aims to ensure that merging is only performed in the direction in which the mesh spacing is smallest. In effect, this means that merging in anisotropic regions is only performed in the directions in which the edge coefficients are large. The condition used at present is to merge an edge if the norm of the edge coefficient is larger than a directionality factor K_d of the average edge weights connected to the seed node, I, i.e. if

$$||C_j^{IJ}|| > K_d \frac{1}{|\Lambda_I|} \sum_{J \in \Lambda_I} ||C_j^{IJ}||\,, \tag{5.55}$$

where $|\Lambda_I|$ is the number of edges connected to the seed. Usually the value of the directionality factor is taken as $K_d = 0.5$. An illustration of the directional agglomeration approach is shown schematically in Figure 5.6(d). When this approach is used for practical problems using hybrid meshes, the coarsening process works correctly, as illustrated in Figure 5.8 for mesh levels one and three. However, it was found that this directional agglomeration scheme did not significantly improve the convergence rate of the scheme. This is thought to be due to less aggressive agglomeration resulting from the reduced connectivities in hybrid meshes. Therefore, for the examples shown in this report, no directional agglomeration was used.

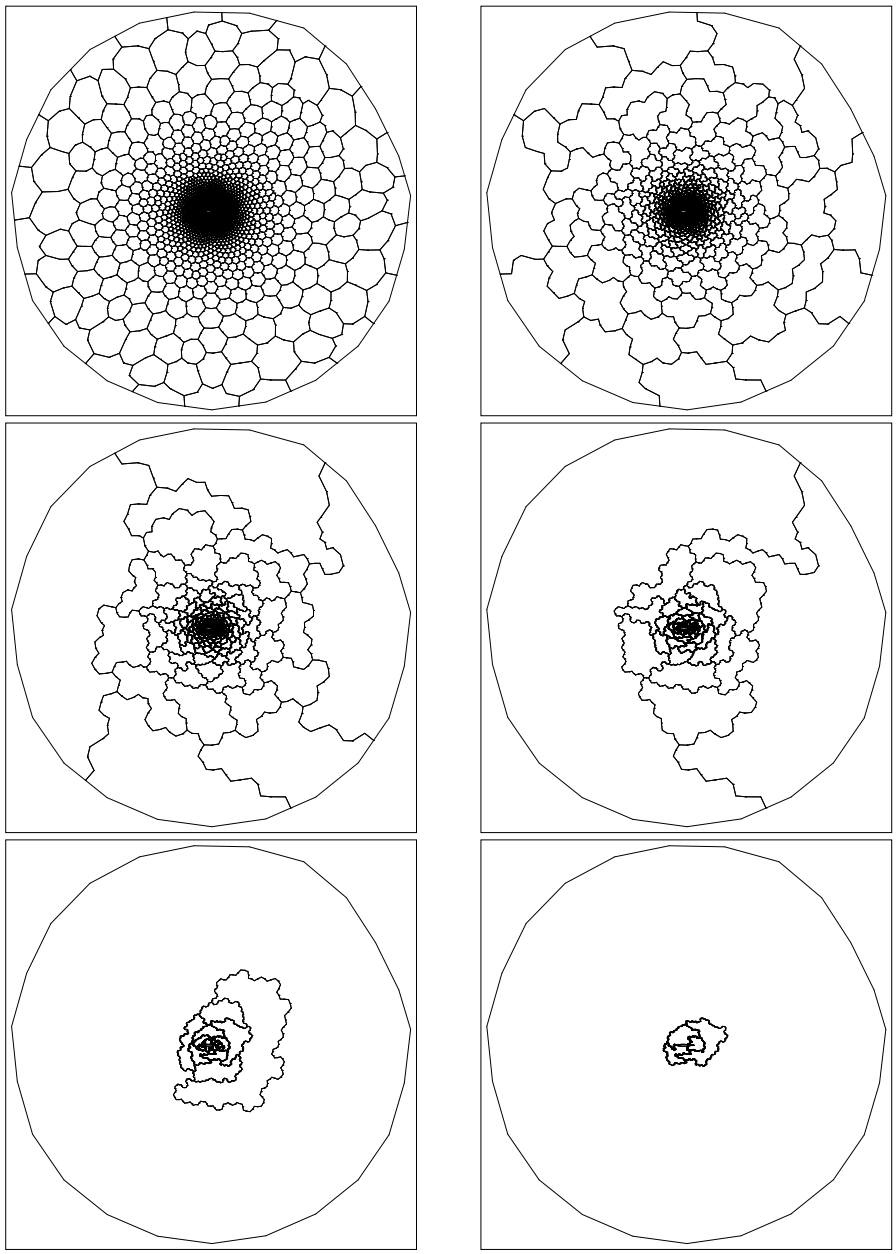

Figure 5.7: Illustration of the homogenous agglomeration procedure showing the control volumes of six grid levels for an aerofoil mesh.

5.5 Time–Dependent Solution Procedures

The time–dependent governing equations are traditionally solved by explicit time stepping. Such schemes are easy to implement and require little extra storage. However, the time step is governed by the stability limit of the smallest cell in the mesh, and this may be quite small compared to the time step required for time–accuracy. This restriction is especially severe for viscous calculations where the allowed time step in the wall layers is very small. The number of iterations

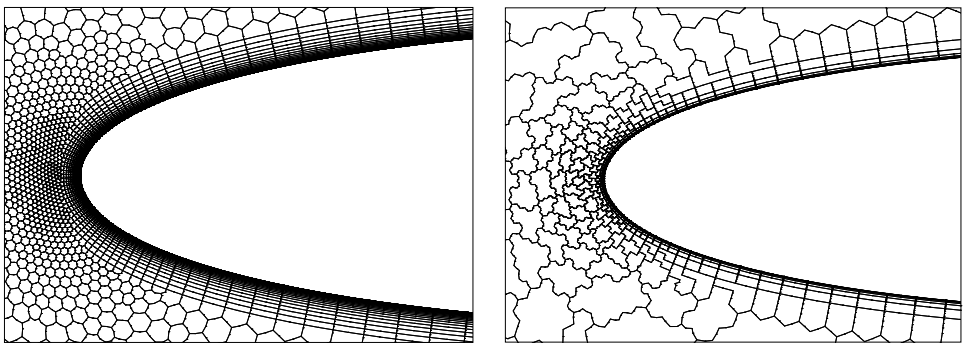

Figure 5.8: Illustration of directional agglomeration on an aerofoil mesh.

needed to solve for the required physical time interval is therefore often excessive. Since the computational effort becomes directly proportional to the physical time interval, explicit time stepping is particularly slow for low frequency cases. It is possible to reduce the computational cost of explicit time stepping schemes by splitting the computational domain into regions according to the local time step [35, 52]. In this way, regions with large allowable time steps are not solved as frequently as the regions where the stability time step is small. Even with this approach, the time required for solving complex problems may be excessive however.

The method used here treats the time term in the governing equations as a source and in this way solves the time dependent equations implicitly. For the scheme of equation (4.84), the terms in equation (5.1) take on the form

$$\begin{aligned} V_I^n R_i^I(\boldsymbol{U}^n) \equiv & \frac{3}{2\Delta t} V_I^n U_i^{I,n} \\ & + \sum_{J\in\Lambda_I} \frac{C_j^{IJ,n}}{2}\left(F_{ij}^{I,n} + F_{ij}^{J,n}\right) + \sum_{J\in\Lambda_I^B} D_j^{IJ,n} F_{ij}^{I,n} \\ & - \sum_{J\in\Lambda_I} \frac{\hat{S}_{IJ}^n}{2}\left(U_i^{I,n} + U_i^{J,n}\right) - \sum_{J\in\Lambda_I^B} \hat{T}_{IJ}^n U_i^{I,n} - \sum_{J\in\Lambda_I} C_j^{IJ} G_{ij}^{IJ} - \sum_{J\in\Lambda_I^B} D_j^{IJ} G_{ij}^I \\ & - \sum_{J\in\Lambda_I} M_{ij}^{IJ,n}\left(E_j^{J,n} - E_j^{I,n}\right) - \sum_{J\in\Lambda_I} \alpha_{IJ} N_{ij}^{IJ,n}\left(U_j^{J,n} - U_j^{I,n}\right) \end{aligned} \tag{5.56}$$

for the discrete operator, and

$$V_I^n S_i^{I,n} \equiv \frac{1}{\Delta t}\left(2V_I^{n-1}U_i^{I,n-1} - \frac{1}{2}V_I^{n-2}U_i^{I,n-2}\right) \tag{5.57}$$

for the source. This approach can be thought of as converging the set of steady state equations, with the addition of the time source, for every physical time step. In this way, there are no numerical limitations of the physical time step, and it can be set to a value governed by accuracy concerns only. Also, since the way the subiterations of each physical time step are conducted is not related to

the accuracy of the scheme, convergence acceleration techniques can be applied. Following this approach, the multigrid scheme used for steady–state calculations can thus be utilized in much the same way for time–accurate calculations, with Runge–Kutta relaxation and local time stepping. The number of physical time steps needed for a calculation can be of two or more orders of magnitude less than what is required if an explicit time stepping scheme is used. The speedup of the multigrid accelerated implicit scheme over explicit time stepping is dependent on the Strouhal number of the flow. Large Strouhal numbers generally favor the implicit approach. Typically, the speedup achieved with the implicit approach is at least one order of magnitude, for minimal additional memory cost.

5.6 Parallelization

The use of parallel computers offers the possibility of solving problems of sizes not readily handled by machines with only a single CPU. For distributed memory machines, the size of problems that may be undertaken on the computer can be increased by letting each node handle a smaller part of the problem. Perhaps more significantly, the time required for solving the problem can be appreciably reduced. This is achieved not only by spreading the workload over several CPU's, but also by reducing the amount of memory each CPU needs to access which often results in more efficient memory handling. In some cases superlinear speedup can be achieved due to this effect.

Since communication of a variable between processors is several orders of magnitude slower than a basic floating–point operation on most computers, efforts must be made to reduce communication when parallelizing a code. Some numerical schemes, such as direct Gauss elimination, are inherently sequential by nature. Such schemes must be altered, by for example domain decomposition methods, to take advantage of the capabilities of a parallel machine. Iterative solution methods on the other hand, are fairly easy to parallelize. This is particularly true for Runge–Kutta schemes, where each node has a very local region of dependency and no matrices are involved.

In the parallelization approach used here, the mesh is either created sequentially and then split into the desired number of computational domains, or generated in parallel using a parallel mesh generator [86]. The domain splitting is performed using the METIS library [43], which employs a multilevel graph partitioning strategy. This approach is very fast, produces well balanced domains and accepts edge–based data structures. METIS does however include a significant memory overhead, typically of the order of 50% of the sequential preprocessor. The domain splitting procedure essentially colours the nodes in the mesh according to which domain they are to belong, Figure 5.9. The preprocessor then renumbers the nodes in the domains and generates the communication data required. The edges of the global mesh are placed in the domain of the nodes attached to the edge, or if the nodes belong to different domains, i.e an *inter–domain edge*, the edge is placed in the domain with the smallest domain number. The edges are then renumbered in each domain, placing the inter–domain edges first in the list. On the boundaries of the domains, the nodes are duplicated so

that each domain stores a copy of the nodes appearing in the edge list. In this way, *ghost nodes* are introduced which do not belong to the domain in question but are allocated a place in memory to store the contributions to these nodes stemming from the current domain. If the node belongs to the current domain, its is termed a *real node*. The communication data structure consists of two registers for each domain combination on each grid level. These registers are termed the *ghost node register* and *real node register* and simply store the local node indexes of the ghost and real nodes respectively. The data structure is illustrated in Figure 5.10. A data structure for the boundary edges is created in an analogous way.

The solver code is essentially built up of loops over edges and loops over nodes. The parallelization procedure can, therefore, be described by just considering these two loops. The edge loops appear, as described in Appendix **??**, wherever boundary integrals are performed. The parallel version of this loop can be summarized as follows:

1. The increment from the current domain on the communication nodes is calculated by looping over the internal and boundary edges that are connected to ghost nodes, Figure 5.11(a).

2. The increments of the ghost nodes in the domain are sent to the associated real nodes using the ghost node register, Figure 5.11(b).

3. The internal and boundary edge increments internal to the domain are calculated, Figure 5.11(c).

4. The ghost increments from the neighbouring domains are received using the real node register and added to the increments of the real nodes. The complete increment has now been found for the real nodes of each domain.

5. The real node register is used to send the node increments to the neighbouring domains with corresponding ghost nodes, Figure 5.11(d).

6. The complete increments are received using the ghost node register and placed in the ghost nodes of the domain, overwriting the previous values of the increment in these nodes.

The approach ensures that the first send is performed as soon as possible, allowing the communication to take place while the edge increments internal to the domain are being calculated.

The node loops usually do not require any communication since the ghost node increments are updated in the edge loop procedure. There are, however, some exceptions where the information stored in the ghost nodes are incorrect and need to be updated from the real node values from the neighbouring domain. A good example of such a case is the intergrid mappings used in the multigrid procedure. The ghost nodes do not receive any contribution from the current domain nodes of another mesh level and need to be updated from the domain where the node is real. This is done by first sending the true values using the real node register and then receiving and placing them in the connected domains using the ghost node register.

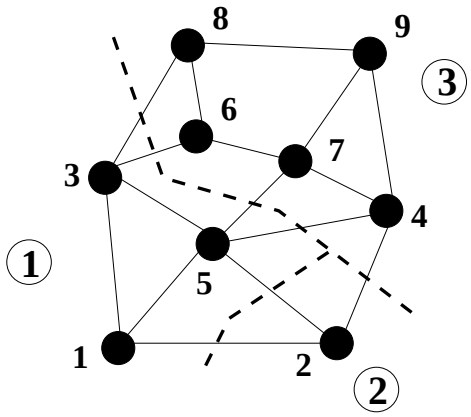

Figure 5.9: Illustration of the mesh partitioning procedure. The boundaries between the domains are shown in dashed bold lines. The global numbering is shown in bold while the domain numbers are shown circled.

Two parallelization approaches were tried in conjunction with the multigrid procedure. The first approach retains the colouring of the fine mesh for the coarse mesh levels. This is incorporated by only allowing merging of control volumes in the same domain in the agglomeration implementation and can thus be termed a *local* agglomeration approach. This means that the parallel multigrid scheme will not be equivalent to a sequential implementation. Of course, since multigrid is consistent, this will only affect the convergence rate of the scheme, not the solution accuracy. It is clear however that this approach may degrade the multigrid performance significantly if a large number of parallel domains relative to the mesh size is used. Also, by using this approach, well balanced coarse meshes can not be guaranteed. This is particularly true if directional agglomeration and/or hybrid meshes are used where it is perfectly possible to have some domains located in the viscous layers, while other domains are situated in the outer mesh. Since the mesh sizes rapidly decrease however, it is likely that the low quality load–balancing will be of relatively little importance on the overall performance of the scheme. The advantages of the approach is that minimum communication is required in the intergrid mappings and that no specific communication registers between the meshes need to be stored.

The second agglomeration approach implemented was a *global* procedure where the agglomeration is performed across the parallel domain boundaries. The coarse grid supernode is defined to belong to the domain of which the majority of its fine mesh nodes are associated. Global agglomeration ensures equivalence with a sequential approach for any number of domains, but the approach increases the communication load in the inter–grid mappings. This effect was, however, found to be negligible since only a few of these mappings need to be performed. The restriction operation using the global procedure can be summarized as follows:

1. Nodes belonging to the current partition on both the fine and coarse grid levels, are mapped in the normal way.

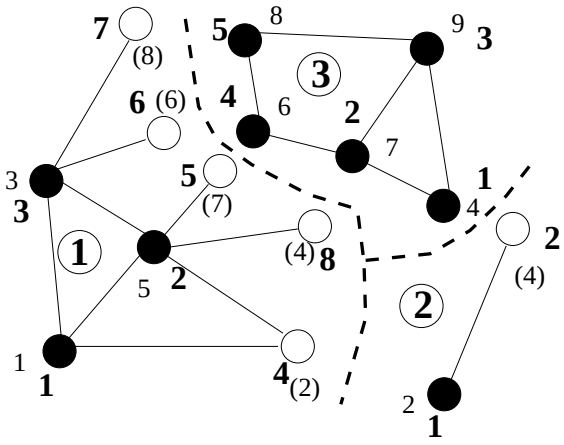

Figure 5.10: Illustration of the datastructure used for the parallelization. The local node numbering is shown in bold and the global numbering is shown using small numbers. The ghost nodes are denoted by the open circles. The global node number to which the ghost node is related is shown in parenreport.

2. The nodes that belong to the current partition on the fine mesh level, but are associated with a coarse mesh supernode of a different domain are mapped into a buffer.

3. The contents of the buffer are transmitted to the other domain using a specific communication array.

4. Contributions to the current domain from other partitions are received and added to the values of the current partition.

5. Information is communicated in the normal way to update ghost nodes.

It can be seen that an additional communication array is needed in the global approach, compared to its local counterpart. These arrays are however small and do not significantly increase the memory requirements of the approach. The prolongation operation is performed as follows in the global strategy:

1. The fine–grid nodes that belong to the current coarse mesh domain are given the associated coarse mesh value in the normal way.

2. The values belonging to the fine mesh nodes associated with a different domain are communicated to the coarse mesh domain of which the fine–grid nodes belong

3. The fine mesh node values belonging to the current domain on the fine mesh level, but are associated with a different domain on the coarse mesh, are received and placed in a buffer.

4. The buffer is communicated to the fine grid level where the values are inserted into the appropriate nodes.

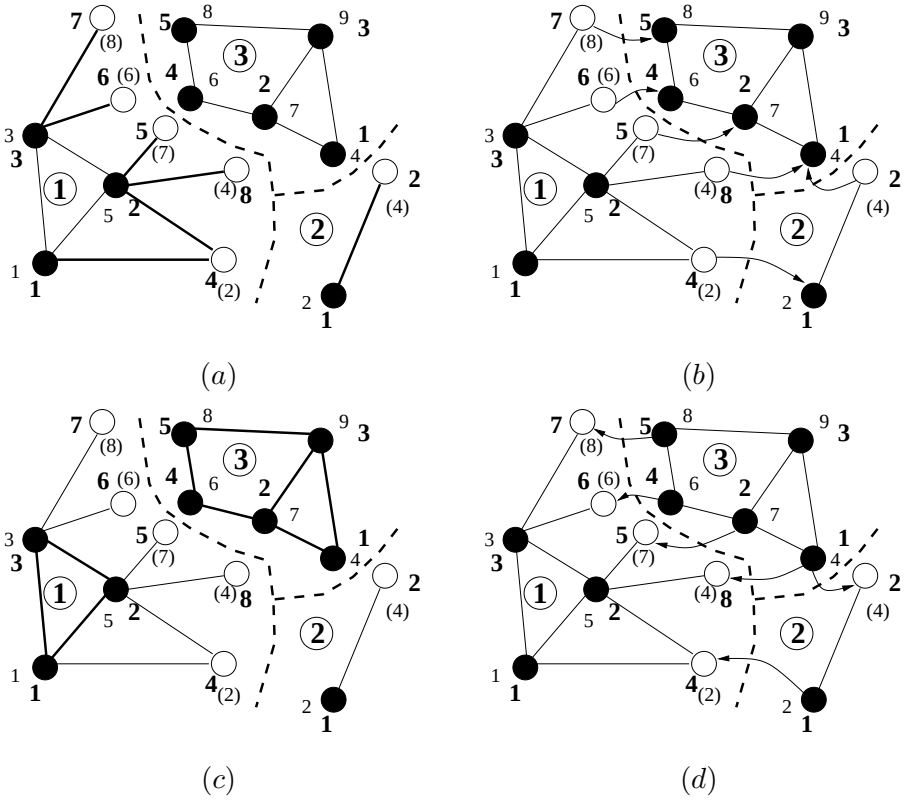

Figure 5.11: Illustration of the parallelization procedure.

5. The ghost nodes are updated in the normal way.

The same communication arrays used for the restriction operator can be used for the prolongation.

While not being able to guarantee good load–balancing on the coarse meshes, the quality of the coarse meshes is, in many cases, significantly better than what is obtained from local agglomeration. This results in fewer cycles for convergence using global agglomeration, with approximately the same cost per cycle. The global agglomeration procedure, while more complicated implementationally, was thus found to be superior to the local procedure and is used in all the examples shown in this report. A comparison between the convergence rates obtained with the two agglomeration approaches on a turbulent wing calculation is shown in Figure 5.12. This is an extreme case where 16 domains were used on a relatively small mesh, and some of the domains generated by the partitioning algorithm were severely warped. The global agglomeration procedure has however consistently outperformed the local approach in all the examples tested.

A third approach for coarse mesh partitioning for parallel solvers is to repartition every mesh level. Using this approach it is, for example, possible to vary the number of partitions on the different mesh levels. The coarse mesh levels can also be properly balanced with this approach. To avoid excessive communication

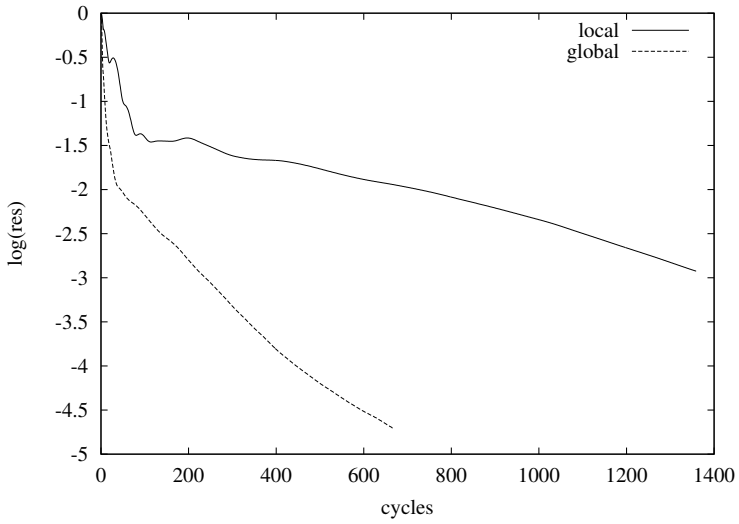

Figure 5.12: Comparison between the convergence rates for the local and global agglomeration procedures for turbulent flow around a wing.

in inter–grid mappings however, it is important that the partitioning of the different mesh levels is somehow connected. The effect of misbalanced coarse mesh domains has, however, proven itself to be relatively small in the calculations performed thus far. The performance of the global agglomeration scheme is therefore considered satisfactory and more advanced parallelization schemes for the coarse meshes in the multigrid procedure are regarded to be beyond the scope of this report.

The parallelization was performed through calls to a local library. The external parallelization library used can thus be easily changed by modifying the local library file only. For the examples shown in this report, the PVM parallelization library [25] was used. Parallelization was only implemented for the solver stage of the solution procedure. The remaining modules are however constructed in a way that should allow for a relatively straightforward parallelization.

Bibliography

[1] S. Allwright. Multiblock topology specification and grid generation for complete aircraft configurations. In *Applications of Mesh Generation to Complex 3-D Configurations, AGARD Conference Proceedings*, volume 464, pages 11.1–11.11, 1990.

[2] B. S. Baldwin and T. J. Barth. A one–equation turbulence transport model for high Reynolds number wall– bounded flows. *NASA Technical Memorandum 102847*, 1990.

[3] B. S. Baldwin and H. Lomax. Thin–layer approximation and algebraic model for separated turbulent flows. *AIAA paper 78-257*, 1978.

[4] J. E. Bardina, P. G. Huang, and T. J. Coaxley. Turbulence modeling validation, testing and develompment. *NASA Technical Memorandum 110448*, 1997.

[5] T. Barth. Randomized multigrid. *AIAA paper 95–0207*, 1995.

[6] J. T. Batina. Unsteady Euler airfoil solutions using unstructured dynamic meshes. *AIAA paper 89–0115, AIAA 27th aerospace sciences meeting*, 1989.

[7] R. M. Beam and R. F. Warming. An implicit finite–difference algorithm for hyperbolic systems in conservation–law form – application to Eulerian gasdynamic equations. *Journal of Computational Physics*, 22:87–110, 1976.

[8] A. Belov, L. Martinelli, and A. Jameson. A new implicit algorithm with multigrid for unsteady incompressible flow calculations. *AIAA paper 95–0049*, 1995.

[9] A. Brandt. Multi–level adaptive solutions to boundary value problems. *Mathematics of Computation*, 21:333–390, 1977.

[10] A. Brandt. *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics*. GMD–Studien Nr. 85, 1984.

[11] J. C. Cavendish, D. A. Field, and W. H. Frey. An approach to automatic three-dimensional finite element mesh generation. *International Journal for Numerical Methods in Engineering*, 21:329–347, 1985.

[12] S. D. Connell and D. G. Holmes. A 3D unstructured adaptive multigrid scheme for the Euler equations. *AIAA paper 93–3339*, 1993.

[13] S. A. Coons. Surfaces for computed aided design of space forms. In *Report MAC–TR–41, Project MAC, M.I.T.*, 1967.

[14] P. I. Crumpton and M. B. Giles. Implicit time–accurate solutions on unstructured dynamic grids. *International Journal for Numerical Methods in Fluids*, 25:1285–1300, 1997.

[15] P. I. Crumpton, P. Moinier, and M. B. Giles. An unstructured algorithm for high Reynolds number flows on highly–stretched grids. In *Proceedings of the 10th International Conference on Numerical Methods for Laminar and Turbulent Flow*, 1997.

[16] P. I. Crumpton, P. Moinier, and M. B. Giles. An unstructured algorithm for high Reynolds number flows on highly stretched grids. In C. Taylor and J. T. Cross, editors, *Numerical Methods in Laminar and Turbulent Flow*, pages 561–572. Pineridge Press, 1997.

[17] J. Donéa. *Arbitrary Lagrangian Eulerian Methods, Computational Methods for Transient analysis*. North–Holland, Elsevier, 1983.

[18] C. Fahrat, C. Degand, B. Koobus, and M. Lesoinne. Torsional springs for two–dimensional dynamic unstructured fluid meshes. *Computational Methods in Applied Mechanical Engineering*, 163:231–245, 1998.

[19] I. D. Faux and M. J. Pratt. *Computational Geometry for Design and Manufacture*. Chichester, 1981.

[20] A. Favre. Equations des gaz turbulents compressibles. *Journal de Mecanique*, 4:361–390, 1965.

[21] R. P. Fedorenko. The speed of convergence of one iterative process. *SSR Computational Mathematics and Mathematical Physics*, 4:227–235, 1964.

[22] J. C. Ferguson. Multivariate curve interpolation. *J. ACM*, 464:221–228, 1964.

[23] J. H. Ferziger. Large eddy numerical simulations of turbulent flows. *AIAA Journal*, 15:1261–1267, 1977.

[24] C. A. J. Fletcher. *Computational Techniques for Fluid Dynamics*, volume II. Springer–Verlag, second edition, 1991.

[25] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing*. The MIT Press, 1994.

[26] A. J. George. *Computer Implementation of the Finite Element Method*. PhD thesis, Stanford University, STAN–CS–71–208, 1971.

[27] P. Geuzaine. *An Implicit Upwind Finite Volume Method for Compressible Turbulent Flows on Unstructured Meshes*. PhD thesis, Université de Liège, 1999.

[28] M. B. Giles. Energy stability analysis of multi–step methods on unstructured meshes. *MIT CFD Laboratory Report, CFDL–TR—87.1*, 1987.

[29] M. B. Giles. Accuracy of node based solutions on irregular meshes. In D. L. Dwoyer, M. Y. Hussaini, and R. G. Voigt, editors, *Proceedings of the 11th International Conference on Numerical Methods in Fluid Dynamics*, pages 272–277. Springer Verlag, 1989.

[30] G. H. Golub and C. F. van Loan. *Matrix Computation*. The John Hopkins University Press, third edition, 1996.

[31] W. Hackbusch. *Multi—Grid Methods and Applications*. Springer–Verlag, 1985.

[32] V. Hannemann. Structured multigrid agglomeration on a data structure for unstructured meshes. In *Numerical Methods for Fluid Dynamics VII, Oxford*, pages 329–336, 2001.

[33] O. Hassan, K. Morgan, E. J. Probert, and J. Peraire. Unstructured tetrahedral mesh generation for three–dimensional viscous flows. *International Journal for Numerical Methods in Engineering*, 39:549–567, 1996.

[34] O. Hassan, E. J. Probert, K. Morgan, and J. Peraire. Line relaxation methods for the solution of 2d and 3d compressible flows. *AIAA paper 93–3366*, 1993.

[35] O. Hassan, E. J. Probert, K. Morgan, and N. P. Weatherill. Unsteady flow simulation using unstructured meshes. *Computer Methods for Applied Mechanical Engineering*, 189:1247–1275, 2000.

[36] O. Hassan, K. A. Sørensen, K. Morgan, and N.P. Weatherill. Unstructured hybrid meshes for the simulation of 3d viscous compressible flow. In *proceedings 7'th International Conference on Numerical Grid Generation in Computational Fluid Simulations, Whistler Canada*, 2000.

[37] C. Hirsch. *Numerical Computation of Internal and External Flows, Computational Methods for Inviscid and Viscous Flows*, volume 2. Wiley, 1990.

[38] Z. U. A. Warsi J. F. Thompson and C. W. Mastin. *Numerical grid generation—Foundations and applications*. North-Holland, 1985.

[39] K. Morgan J. Peraire, M. Vahdati and O. C. Zienkiewicz. Adaptive remeshing for compressible flow computations. *J. Comp. Phys.*, 72:449–466, 1987.

[40] A. Jameson. Time dependent calculations using multigrid, with applications to unsteady flows past aerofoils, wings and helicopter rotors. *AIAA paper 91–1596*, 1991.

[41] A. Jameson, W. Schmidt, and E. Turkel. Numerical simulation of the Euler equations by finite volume methods using Runge–Kutta timestepping schemes. *AIAA paper 81–1259, AIAA 5th Computational Fluid Dynamics Conference*, 1981.

[42] G. E. Karniadakis and S. J. Sherwin. *Spectral/hp Element Methods for CFD*. Oxford University Press, 1999.

[43] G. Karypis and V. Kumar. Multilevel k–way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48:96–129, 1998.

[44] J. Kim, P. Moin, and R. Moser. Turbulence statistics in fully developed channel flow at low Reynolds number. *Journal of Fluid Mechanics*, 177:133–166, 1987.

[45] D. D. Knight. *Numerical Simulation of Compressible Turbulent Flows Using the Reynolds–Averaged Navier–Stokes Equations*, chapter 5: Turbulence In Compressible Flows. AGARD R-819, 1997.

[46] A. N. Kolmogorov. Local structure of turbulence in incompressible viscous fluid for very large Reynolds number. *Doklady AN. SSSR*, 30:299–303, 1941.

[47] B. Koobus and C. Fahrat. Second–order time–accurate and geometrically conservative implicit schemes for flow computations on unstructured dynamic meshes. *Computational Methods in Applied Mechanical Engineering*, 170:103–129, 1999.

[48] B. E. Launder and B. I. Sharma. Application of the energy dissipation model of turbulence to the calculation of flow near a spinning disc. *Letters in Heat and Mass Transfer*, 1:131–138, 1974.

[49] C. K. Lee and S. H. Lo. A new scheme for the generation of graded quadrilateral meshes. *Computers and Structures*, 52:847–857, 1994.

[50] D. T. Lee and B. J. Schachter. Two algorithms for constructing a Delaunay triangulation. *International Journal of Computer and Information Sciences*, 9:219–242, 1980.

[51] M. Lesoinne and C. Fahrat. Geometric conservation laws for flow problems with moving boundaries and deformable meshes, and their impact on aerolastic computations. *Computational Methods in Applied Mechanical Engineering*, 134:71–90, 1996.

[52] R. Lőhner, K. Morgan, and O. C. Zienkiewicz. The use of domain splitting with an explicit hyperbolic solver. *Computer methods in Applied Mechanics and Engineering*, 45:313–329, 1984.

[53] R. Lőhner and P. Parikh. Three–dimensional grid generation by the advancing front method. *International Journal of Numerical Methods in Fluids*, 8:1135–1149, 1988.

[54] H. Luo, J. Baum, and R. Lőhner. Edge–based finite element scheme for the Euler equations. *AIAA Journal*, 32:1183–1190, 1994.

[55] H. Luo, J. D. Baum, and R. Lőhner. A fast, matrix–free implicit method for compressible flows on unstructured grids. *Journal of Computational Physics*, 146:664–690, 1998.

[56] H. Luo, D. Sharov, J. D. Baum, and R. Lőhner. On the computation of compressible turbulent flows on unstructured grids. *AIAA paper 00–0926*, 2000.

[57] H. Steve M. H. Lallemand and A. Dervieux. Unstructured multigridding by volume agglomeration: current status. *Computers & Fluids*, 21:397–433, 1992.

[58] R. W. MacCormack and B. S. Baldwin. A numerical method for solving the Navier–Stokes equations with application to shock–boundary layer interactions. *AIAA paper 1–75*, 1975.

[59] M. T. Manzari. *An Unstructured Grid Finite Element Algorithm for Compressible Turbulent Flow Computations*. PhD thesis, University of Wales, Swansea, 1996.

[60] D. J. Mavriplis. On convergence acceleration techniques for unstructured meshes. *ICASE Report No. 98–44*, 1998.

[61] D. J. Mavriplis. Multigrid approaches to non–linear diffusion problems on unstructured meshes. *ICASE Report No. 2001–3, NASA/CR–2001–210660*, 2001.

[62] D. J. Mavriplis and A. Jameson. Multigrid solution of the two–dimensional Euler equations on unstructured triangular meshes. *AIAA Journal*, 26:824–831, 1988.

[63] D. J. Mavriplis and V. Venkatakrishnan. A 3D agglomeration multigrid solver for the Reynolds–averaged Navier–Stokes equations on unstructured meshes. *International Journal for Numerical Methods in Fluids*, 23:527–544, 1996.

[64] F. R. Menter. Eddy viscosity transport equations and their relation to the k–ϵ model. *NASA Technical Memorandum 108854*, 1994.

[65] F. R. Menter. Two–equation eddy viscosity turbulence models for engineering applications. *AIAA Journal*, 4:1299–1310, 1994.

[66] A. Moitra. Unstructured grid issues in 2–d high–lift computations. In B. K. Soni, J. Hauser, J. F. Thompson, and P. Eiseman, editors, *Proceedings of the 7th international conference on numerical grid generation in computational field simulations*, pages 263–274. International society of grid generation, 2000.

[67] E. Morano, H. Guillard, A. Dervieux, M. P. Leclercq, and B. Stoufflet. Faster relaxations for non–structured multigrid with Voronoi coarsening. In *Proceedings of ECCOMAS 1992*, volume 1, pages 69–74, 1992.

[68] K. Morgan and J. Peraire. Unstructured grid finite element methods for fluid mechanics. *Reports of Progress in Physics*, 61:569–638, 1998.

[69] M. V. Morkovin. Effects of compressibility on turbulent flow. In A. Favre, editor, *The Mechanics of Turbulence*. Gordon and Breach, 1962.

[70] W. A. Mulder. A high–resolution Euler solver based on multigrid, semi–coarsening, and defect correction. *Journal of Computational Physics*, 100:91–104, 1992.

[71] J.D. Muller and M.B. Giles. Edge–based multigrid schemes for hybrid grids. In *Proceedings of 6th ICFD Conference on Numerical Methods for Fluid Dynamics, Oxford*, 1998.

[72] K. Nakahashi. FDM–FEM zonal approach for viscous flow computations over multiple bodies. *AIAA paper 87–0604*, 1987.

[73] B. Nkonga and H. Guillard. Godunov type method on non–structured meshes for three–dimensional moving boundary problems. *Computational Methods in Applied Mechanical Engineering*, 113:183–204, 1994.

[74] J. Peiró. *A finite element procedure for the solution of the Euler equations using unstructured meshes*. PhD thesis, University of Wales, Swansea, 1989.

[75] J. Peraire, J. Peiró, L. Formaggia, K. Morgan, and O. C. Zienkiewicz. Finite element Euler computations in three dimensions. *International Journal of Numerical Methods in Engineering*, 26:2135–2159, 1988.

[76] J. Peraire, J. Peiró, and K. Morgan. Finite element multigrid solution of Euler flows past installed aero–engines. *Computational Mechanics*, 11:433–451, 1993.

[77] E. Perez. A 3D finite–element multigrid solver for the Euler equations. *INRIA Report No. 442*, 1985.

[78] N. A. Pierce and M. B. Giles. Preconditioned multigrid methods for compressible flow calculations on streched meshes. *Journal of Computational Physics*, 136:425–445, 1997.

[79] M. M. Rai and P. Moin. Direct simulations of turbulent flow using finite differences. *Journal of Computational Physics*, 96:15–53, 1991.

[80] B. A. Petterson Reif. Aspects of algebraic modelling for complex turbulent flows. In *Proceedings of the 13th nordic seminar on computational mechanics*, pages 44–47. Unipub forlag, 2000.

[81] A. A. G. Requicha and H. B. Voelcher. Solid modelling: A historical summary and contemporary assessment. *IEEE Computer Graphics and Applications*, 73:9–24, 1982.

[82] P. L. Roe. Approximate Riemann solvers, parameter vectors and difference schemes. *Journal of Computational Physics*, 63:458–476, 1981.

[83] Y. Saad. Preconditioning techniques for nonsymmetric and indefinite linear systems. *Journal of Computational and Applied Mathematics*, 24:89–105, 1988.

[84] Y. Saad and M. H. Schultz. GMRES: A generalised minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 7:856–869, 1986.

[85] V. A. Sai and F. M. Lutfy. Analysis of the Baldwin–Barth and Spalart–Allmaras one–equation turbulence models. *AIAA Journal*, 33:1971–1974, 1995.

[86] R. Said, N. P. Weatherill, K. Morgan, and N. A. Verhoeven. Distributed parallel Delaunay mesh generation. *Computer Methods in Applied Mechanics and Engineering*, 177:109–125, 1999.

[87] A. M. O. Smith and T. Cebeci. Numerical solution of the turbulent boundary–layer equations. *Douglas Aircraft Division Report DAC 33735*, 1967.

[88] P.R. Spalart and S.R. Allmaras. A one–equation turbulent model for aerodynamic flows. *AIAA paper 92–0439*, 1992.

[89] C. G. Speziale, R. Abid, and E. C. Anderson. A critical evaluation of two–equation models for near wall turbulence. *AIAA paper 90–1481*, 1990.

[90] R. C. Swanson and E. Turkel. Multistage schemes with multigrid for Euler and Navier–Stokes equations, components and analysis. *NASA Techical Paper 3631*, 1997.

[91] H. Tennekes and J. L. Lumley. *A First Course in Turbulence*. The MIT Press, 1972.

[92] T. Tezduyar, M. Behr, and J. Liou. A new strategy for finite element computations involving moving boundaries and interfaces– the deforming spatial domain/space–time procedure: I. The concept and the preliminary numerical testcases. *Computer Methods in Applied Mechanical Engineering*, 94:339–351, 1992.

[93] P. D. Thomas and C. K. Lombard. Geometric conservation law and its application to flow computations on moving grids. *AIAA Journal*, 17:1030–1037, 1979.

[94] W. J. Usab and E. M. Murman. Embedded mesh solutions of the Euler equations using a multiple–grid method. In W. G. Habashi, editor, *Advances in Transonics*. Pineridge PRess, Swansea, 1985.

[95] M. Vahdati, K. Morgan, J. Peraire, and O. Hassan. A cell–vertex upwind unstructured grid solution procedure for high–speed compressible viscous flow. In *proceedings International Conference of Hypersonic Aerodynamics, Roayal Aeronautical Society, London*, 1989.

[96] V. Venkatakrishnan and D. J. Mavriplis. Implicit solvers for unstructured meshes. *Journal of Computational Physics*, 105:83–91, 1993.

[97] V. Venkatakrishnan and D. J. Mavriplis. Implicit method for the computation of unsteady flows on unstructured grids. *Journal of Computational Physics*, 127:380–397, 1996.

[98] J. von Neumann and R. D. Richtmyer. A method for the numerical calculations of hydrodynamical shocks. *Journal of Mathematical Physics*, 21:232–237, 1950.

[99] S. Ward and Y. Kallinderis. Hybrid prismatic/terahedral grid generation for complex 3-d geometries. *AIAA paper 93–0669*, 1993.

[100] N. P. Weatherill and O. Hassan. Efficient three–dimensional Delaunay triangulation with automatic boundary point creation and imposed boundary constraints. *International Journal for Numerical Methods in Engineering*, 37:2005–2039, 1994.

[101] N. P. Weatherill, O. Hassan, and D. L. Marcum. Calculation of steady compressible flowfields with the finite element method. *AIAA paper 93-0341*, 1993.

[102] N. P. Weatherill, O. Hassan, and D. L. Marcum. Compressible flowfield solutions with unstructured grids generated by Delaunay triangulation. *AIAA Journal*, 33:1196–1204, 1995.

[103] D. C. Wilcox. *Turbulence Modeling for CFD.* DCW Industries, 1993.

[104] D. C. Wilcox and I. E. Alber. A turbulence model for high speed flows. In *Proceedings of the 1972 Heat Transfer & Fluid Mechanics Inst.*, pages 231–252. Stanford University Press, 1972.

[105] J. S. Wong, D. L. Darmofal, and J. Peraire. High–order finite element driscretization for the compressible Euler and Navier–Stokes equations. *FDRL TR-01-1 Department of Aeronautics and Astronautics, MIT*, 2001.

[106] O. C. Zienkiewicz, P. Nithiarasu, R. Codina, M. Vasquez, and P. Ortiz. The characteristic–based–split procedure: An efficient and accurate algorithm for fluid problems. *International Journal for Numerical Methods in Fluids*, 31:359–392, 1999.