Mesh generation for large-scale and complex computational simulation.

Larwood, Benjamin Guy

School of Engineering
University of Wales Swansea

MESH GENERATION FOR LARGE-SCALE AND COMPLEX COMPUTATIONAL SIMULATION

BENJAMIN GUY LARWOOD
BEng (Hons), MSc

Thesis submitted to the University of Wales in candidature for the degree of
Doctor of Philosophy

April 2003

Declaration

This work has not previously been accepted in substance for any degree and is not concurrently submitted in candidature for any degree.

Signed .... (Candidate)

Date ....17-7-03.................................

Statement 1

This thesis is the result of my own work/investigation except where otherwise stated. Other sources have been acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed .... (Candidate)

Date ....17-7-03.................................

Statement 2

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loans, and for the title and summary to be made available to outside organisations.

Signed .... (Candidate)

Date ....17-7-03.................................

ACKNOWLEDGEMENTS

I would like to express my gratitude to Nigel Weatherill for giving me the chance to study for this PhD. The encouragement and guidance that I received from Nigel, Oubay Hassan and Ken Morgan made the time that I spent at Swansea University enjoyable.

My partner Jeni Batchelor deserves a special mention for the emotional support throughout the long days. I would like to thanks my parents, Allan and Carolyn, and my sister Lucy, for always being at the end of a telephone line for moral support.

My thanks also go to my friends in Swansea, Jason Jones, James MacFadden, Matthew Profit, Dan van der Leer and at home who provided support in times of debugging despair. Finally to Lynette and Anne, for just being there.
(Something like that would do nicely)

SUMMARY

This thesis presents work in the area of mesh generation for large scale and complex computational simulation. The work covers two areas of great interest within the field of mesh generation; anisotropic mesh generation and parallel large scale mesh generation. Examples of anisotropic Delaunay mesh generation are presented with application to fluid dynamics and computational electromagnetic scattering simulations. Results are shown with reference to simulation accuracy and computational efficiency.

Research into parallel mesh generation is presented and a method of parallel Delaunay mesh generation suitable for use on distributed and shared memory parallel computers is described. Results are shown with reference to computational efficiency, memory usage and finale mesh quality. Examples of meshes generated in parallel are shown for both computational fluid dynamics simulations on simple aeronautical geometries to full aircraft and computational electromagnetic scattering simulations on full aircraft. The meshes range in size from a few thousand tetrahedral elements to a mesh for a computational electromagnetic simulation containing approximately one billion tetrahedral elements.

Table of Contents

Table of Figures

1 Introduction

Computational simulation for engineering applications has been used to reduce development costs for over two decades. It is used in most branches of engineering as an essential tool to reduce development time. The understanding of physical processes is also improved by the use of computational simulation, since complex behaviour can be closely scrutinised.

Research into computational simulation aims toward a common goal: to improve solution accuracy and to utilise new developments and technologies. Providing these tools with user-friendly interfaces can only promote the use of computational simulation within industry.

The finite element method provides a means with which physical problems can be simulated. Initially developed for civil engineering applications, the method involves determining the solution locally on smaller sub-problems, called elements, which combine to form the overall solution. The method has many applications, including solid mechanics, soil mechanics, fluid flow and electromagnetic scattering simulations [1], and stands alongside other methods such as finite difference and finite volume methods [2] to determine the solution to sets of differential equations that describe the physical behaviour of the problem. The accuracy of the solution determined by solving the differential equations that govern the physical process is reliant on both the numerical formulation and the discretisation of the domain. An exact solution of the physical process can only be obtained by using an infinite number of elements, depending also on the consistency of the numerical formulation. Hence, an approximation is used, that forms a compromise between computational expense (the size of the discretisation and numerical formulation) and the accuracy of the simulation. The discretisation of the computational domain into smaller domains that cover the domain in its entirety is the second stage in the simulation process, and forms an important part of this process. The process consists of five stages, shown in Figure 1.

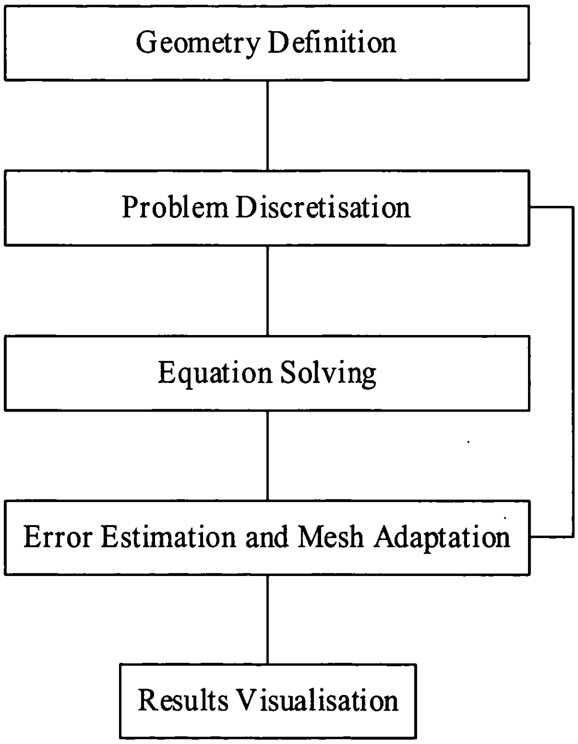


Figure 1 Simulation process

The first stage in the simulation cycle is the geometry definition. Known also as problem definition, this involves producing a computational model that accurately represents the physical problem to be modelled. Typically, this model is derived from designs produced using CAD tools such as CATIA[3] and CADDS[4]. A valid computational geometry is commonly referred to as a "water-tight" geometry. This term signifies that the domain is closed, with neither overlapping surfaces nor gaps between surfaces. Obtaining a model that does not exhibit these features is a research area in it's own right, and is referred to as CAD repair. Once a valid computational model has been derived from the input data, the second stage can begin. During the second stage, the model is sub-divided into smaller regions, known as elements. The collection of elements that fill the computational domain is known as a grid or mesh, the two terms being interchangeable. The third stage involves determining the approximate solution of the numerical problem by solving the finite element equations, which replace the differential equations that govern the physics of the simulation. Stage four is not always used for simulations, but consists of determining the error induced by the discretisation or numerical formulation. The mesh can be adapted to increase the point density (the number of points) in regions of high error, in an attempt to reduce the error in the solution. The solution is an approximation, and is analogous to describing a curve with straight lines. The tighter the curve, the more straight-line sections are required to approximate the curve. The distance that the set of straight lines are away from the exact curved line is the error induced by the approximation. This analogy is shown in Figure 2, where the exact curve is drawn with a red line, and the approximation with straight edges is drawn in black.

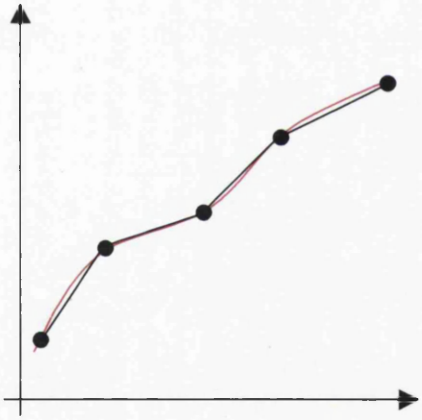


Figure 2 Analogy of curve fitting

The final stage is solution interrogation, where regions of interest can be visualised and numerical data extracted from the solution. The first two stages are grouped together and known collectively as the pre-processing stage. A pre-processor would have as its input a geometric model, and output would be the mesh suitable for use in the third stage, known as the processing stage. Consequently, the visualisation and results interrogation stage is known as post-processing.

A mesh is required to exhibit a number of features that can have a distinct effect on the simulation result. The size of the mesh must be controlled, and user definition of the element size is required in order to resolve high gradient in the solution, and to reduce errors. Thus user knowledge is a factor of the simulation; this must be of both the physics that are being modelled and of the mesh generation and solution algorithms. The quality of the elements forming the mesh is also critical for obtaining accurate solutions and maximising computational efficiency. Fluid flow and electromagnetic scattering solvers are dependant on the quality of the mesh, where flat and badly formed elements can introduce spurious results into the final solution.

A mesh generator must therefore fulfil a distinct set of requirements to be useful to the engineering community:

- Ease of use – the input to the generator must be easily understandable. The controls easily used and effective, and meaningful messages should be produced should any errors occur
- Automatic – as far as possible, the mesh generator should be automatic
- Quality – the mesh generated must exhibit good quality indicators to optimise the solution stage
- Reproducibility – The results from a mesh generator should be reproducible. Any differences due to numerical instabilities should be minimised
- Robust – The mesh generator should be robust in its implementation and the underlying numerical scheme, with good error handling
- Simulation Independent – The mesh generator should be able to provide meshes that are suitable for a wide range of problems, with any extensions (such as

anisotropic elements for the boundary layer of viscous fluid flow simulations) easily accessible.

In aerospace engineering, the cost of developing and testing a new design are extremely high. Traditionally, aircraft have been designed and tested using wind tunnels, with scaled models initially, then moving to full-scale mock-ups. Wind tunnels can cost as much as £4000 per day for a full-scale low speed wind tunnel, in addition to the associated costs of model creation. Taking this into consideration computer simulation becomes an attractive alternative to prolonged wind tunnel testing, enabling the aerospace designer the ability to test designs and developments quickly.

In the early 1970's computing power reached levels with which it was possible to solve the non-linear Navier-Stokes equations, in simplified Euler form, in a reasonable amount of time on two dimensional geometries. Computational fluid dynamics (CFD) became a fast moving research area, driven by intense interest from industry. Research efforts across the world coupled with the ever increasing computer power have allowed fluid flow simulations to become more accurate, modelling complex fluid flow situations. As the simulations became more complex, in terms of geometry and solution algorithm complexity, the solution time increased. This increased complexity places demands on mesh generators to generate meshes of a suitable size capable of resolving the complex features, and generating meshes containing suitably shaped elements. The research presented within this thesis covers aspects of mesh generation related to complex fluid flow and electromagnetic scattering simulations. As the simulation complexity increases, in terms of the physics and geometry definition, the size of mesh require to resolve the solution to the required degree of accuracy increases. Typical figures for mesh size in comparison with the flow equations that are being solved are shown in Figure 3.

Simulation Type	**Typical number of nodes required**
Euler	10^6
Laminar	10^7
Turbulent	10^8
LES	10^{11}

Figure 3 Typical mesh sizes for resolving different flow features on an aerospace geometry

It is clear then that the large meshes are required in order to reduce errors to acceptable levels using current flow solver technology. Hence, mesh generators must be developed that can model geometries accurately, whilst retaining the requirements as described previously. Generating large meshes represents a considerable challenge for mesh generators using sequential methods. High power parallel computers formed from networks of workstations provide large computing power for the processing stage of a simulation, where the mesh is decomposed into a number of smaller sub-domains, and distributed amongst the computers available. Using parallel mesh decomposition tools, allows the sub-domains to be balanced in terms of workload (number of elements per processor) and communication cost. However, the generation of meshes for these simulations remains a sequential task, and therefore requires a computer with significant amount of memory available to a single processor. Presented in Section 4 is work into developing a stable parallel mesh generator implementation of a Delaunay based method. The tool developed consists of fully automatic domain decomposition to allow generation of the mesh across parallel platforms. Included in the tool is the ability to provide a region of stretched elements that has been used in high Reynolds flow simulations to reduce the number of elements in the mesh whilst retaining solution accuracy, and also used in electromagnetic simulations once again to reduce the size of the mesh required.

Section 3 presents work in developing fully automatic anisotropic mesh generation and adaptation methods. Concentrating in two-dimensions, the work is validated by a number of testcases. Anisotropic or stretched elements can be used to reduce the size of a discretisation whilst retaining the solution accuracy of the larger mesh.

2 Sequential Mesh Generation Methods

Two classifications of mesh generation methods exist, structured and unstructured. The difference between a mesh generated using these techniques is the manner in which the neighbours of a point are defined. Structured meshes have a strict order, in that a neighbouring point is easily identifiable, Figure 4. The nodes of a structured mesh are typically numbered in the finite difference numbering scheme {i,j}, in order to find the neighbouring point to a given point P(i,j), increasing or decreasing in value one of the indices will yield the neighbour.

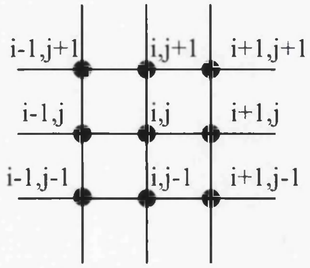


Figure 4 Numbering scheme of a structured mesh

However, unstructured meshes do not have this constraint of neighbouring nodes applied to them. The nodes that form an unstructured mesh are numbered in a contiguous single number format. Element connectivity must be defined explicitly using a connectivity table, which contains for each element within the mesh the node numbers that form the element. Figure 5 shows an unstructured mesh of a simple circular domain. The nature of the mesh is clearly evident, where the neighbours of a point are not clearly identifiable.

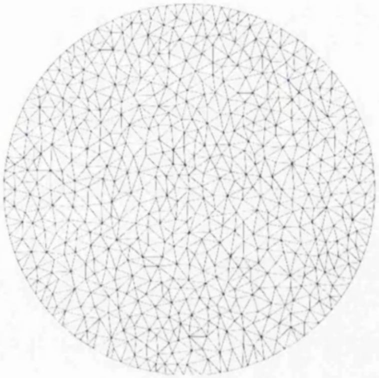


Figure 5 Unstructured mesh example

Structured meshes typically consist of quadrilateral or hexahedral elements, with the generation of simplexes (triangles or tetrahedra) by dissection of these elements. Figure 6 shows a structured mesh of a circular domain. This mesh is known as a type 2 structured mesh, where the structure of the mesh has been defined to avoid degenerate or highly skewed elements. Unstructured mesh generation methods, however, typically produce simplex elements (triangles in two-dimensions, tetrahedra in three-dimensions), and allow complex geometries to be meshed in a more automatic manner than structured methods.

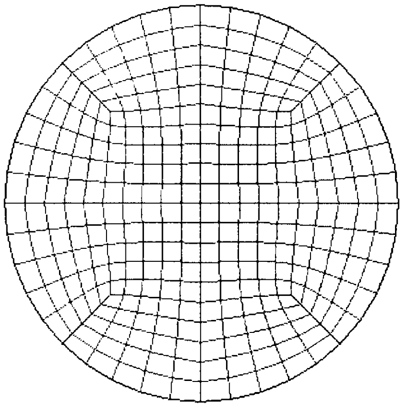


Figure 6 Structured mesh example

The following is a brief overview of the meshing methods that have been developed to provide a basis for finite element simulations.

2.1 Structured Mesh Generation Methods

All the methods that are used to generate structured meshes do so by generating a mesh within a transformed rectangular computational domain, which is mapped into the physical domain to form the mesh of the geometry. Two sub-classes of structured meshing techniques exist; algebraic based methods [5][8] and partial differential equation based methods [5][8]. Structured meshes were recognised as a tool to discretise the physical region for computational fluid dynamics applications, where the numerical algorithms can require the mesh points to follow approximately the flow.

The algebraic based methods consist of interpolation schemes. Here, the boundary values are interpolated into the interior of the domain to produce the required number of points within the domain. Various interpolation techniques have been developed, such as Lagrange interpolation and Hermite interpolation. More commonly, the method of transfinite interpolation[6] is used.

The generation of the interior points in the physical space from a boundary description has been recognised as a boundary value problem [5]. The solution of boundary value problems is one of partial differential equations and the solution of the systems formed by these equations. Elliptic partial differential equations can be used where the domain is closed, where the solution at the boundary is of interest, such as solid mechanics problems. Hyperbolic or parabolic equations are normally used where the domain is open, where the solution at the outer boundary is of little interest, such as fluid dynamic simulations over aerofoils. Using these equations, controls can be applied to control the mesh spacing in defined regions in order to capture the solution accurately.

In order to discretise complex geometries, multiblock techniques were developed. Dividing the physical domain into smaller sub-domains that can then be discretised using partial differential equation or interpolation based methods. Careful attention must be paid to the inter-domain regions, to ensure continuity of elements between neighbouring blocks. Figure 7 shows a typical multiblock mesh for a solid mechanics simulation. The separate blocks are created and meshed individually, by setting the boundary points, and then using the generation methods described previously to determine the interior point positions. These blocks of meshes are then attached to others parts, combining to form the full geometry.

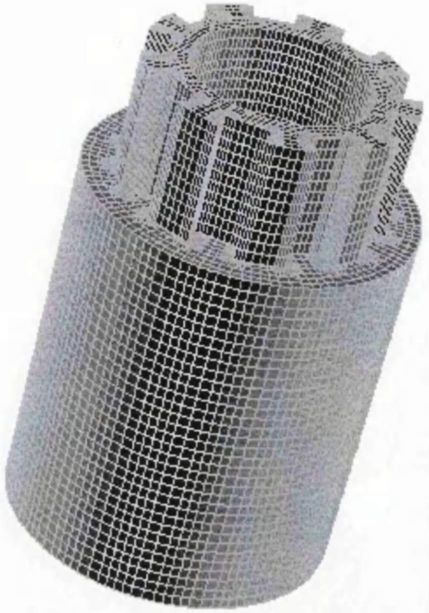


Figure 7 Multiblock mesh example

The work in structured meshing has been written in many texts [5][7][8] and presented at many conferences, such as the conference series "Numerical Grid Generation in Computational Field Simulation" and the Meshing Roundtable Conference Series. The reader is directed to these references and conference proceedings for a more detailed explanation in the subject of structured mesh generation.

2.2 Unstructured Mesh Generation Methods

Unstructured mesh generation techniques grew from a requirement to discretise complex geometries in short time scales. Whereas structured mesh methods have their base in the mathematical concepts of differential equations, unstructured mesh methods come from a geometrical basis. Due to this basis, the methods are automatic in nature, requiring little user intervention even when discretising complex geometrical features. Two methods for unstructured mesh generation are commonly used, the advancing front method and Delaunay based methods. These methods differ by the manner in which points and elements are introduced into the computational domain.

The advancing front technique was first described by A. George [9]. In two dimensions, the scheme starts from a discretisation of the boundary. Figure 8 shows a simple geometry discretised by N points. Starting from the shortest edge (an element consisting of two points), a new point is introduced into the unmeshed domain space. The position of this new point is determined from background mesh and mesh spacing control functions. In order to create a new element, both nodes of the original edge are connected to the new node, and the element connectivity added to the mesh connectivity table. To ensure a valid mesh, this element must be checked for intersection with the existing elements in the mesh. More information regarding the advancing front technique is shown amongst other places in [7][8].

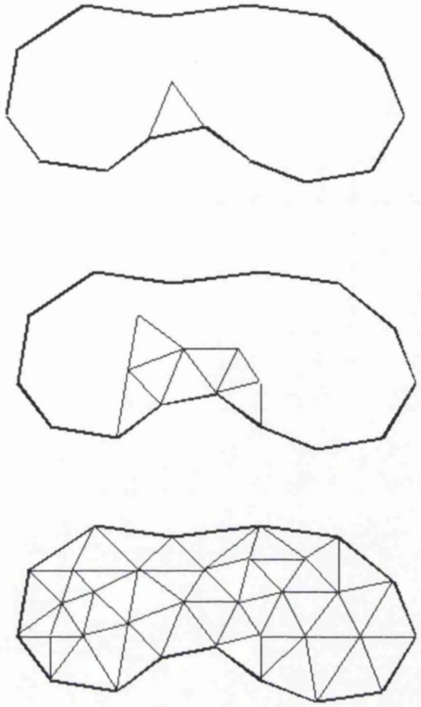


Figure 8 Advancing front method

The advancing front method introduces elements sequentially into the unmeshed space, whereas a Delaunay based scheme inserts points sequentially into the mesh, and determines the new mesh including the new point. The Delaunay method describes a technique of connecting a set of points in such a way that the circumcircle of each element (described by the three nodes) does not contain any other node[38]. A Delaunay mesh is derived from the Voronoi regions [37], which are the regions that are closer to each node than any other. The connection of the Voronoi regions defines a set of tiles known as a Dirichlet tessellation [36]. Given a set of nodes, the Voronoi regions are defined by the lines (bisectors) that mark the equi-distances between the nodes (Figure 9). The connection of the nodes across these bisectors results in the creation of simplexes. Delaunay techniques start from a convex hull, which, by definition, includes all of the points that defined the physical domain boundary. A coarse triangulation of the four points of the convex hull is created, and each point of the boundary is iteratively inserted, creating elements as the method proceeds. Once all points of the boundary have been inserted, a boundary recovery procedure must be used to enforce the boundary edges, which may have been removed. These boundary edges and faces can be lost since the points of the boundary are inserted into the triangulation without consideration for the edges and boundary faces' connectivity. This method of generating a Delaunay satisfying mesh is known as the Bowyer-Watson [10][11] method.

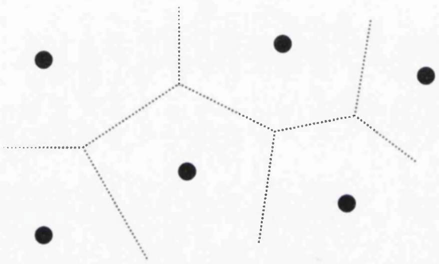


Figure 9 Voronoi Regions of a set of nodes

Other methods for discretising the domain have been developed. Yerry and Shephard [12], introduced the spatial decomposition technique involves applying a quadtree or octree over the domain. Recursive decomposition of the domain until each cell in the tree has reached the element size as defined by the background mesh. This technique results in quadrilaterals across the domain, which can then be sub-divided to form triangles or tetrahedra. The boundary of the domain must be enforced, where the boundary edges are not aligned with the edge of the quadtree. Further references for unstructured meshing techniques are covered in texts such as [7][8] and conferences such as the Numerical Mesh Generation in Computational Field Simulation, Trends in Unstructured Mesh Generation and the Meshing Roundtable series.

2.3 Summary

This chapter has introduced the two types of mesh; structured and unstructured. The methods used to generate these different types of meshes have been described to give the reader a basis for the work to follow. Analysing the two principal schemes, advancing front and Delaunay based methods, it can be shown that due to the intersection checking at each point insertion, advancing front schemes are typically one to two orders of magnitude slower to generate a given number of elements than that of a Delaunay based scheme. For this reason, the extension of the Delaunay based methods to generate meshes that contain many millions of elements required to allow large scale simulations to be performed, is preferred. The following chapters discuss the extension of the Delaunay based Bowyer-Watson method to generate stretched anisotropic elements in order to reduce the mesh size for a required accuracy level, and to extend the method for use on parallel computer architecture.

A survey of meshing programs, available commercially and research codes has been published by Owen[13]. This survey shows that over two-thirds of the products available are unstructured type codes. Of these unstructured mesh generation programs, the programs that produce triangle or tetrahedra use some form of Delaunay algorithm at a ratio of 2:1 compared to advancing front and octree type algorithms. The products include extra features, the most common feature being mesh refinement. Mesh anisotropy, boundary layer definition and adaptivity also feature.

Methods to generate unstructured hexahedra are very desirable, being the focus of intense research [14][15].

3 Delaunay Mesh Generation

The work in unstructured meshing initially concentrated on generating high quality isotropic meshes, where the elements (predominantly triangles and tetrahedra) are close to equilateral. The unique property of a Delaunay triangulation is that given a set of points, a triangulation of these that satisfies the Delaunay criterion for each element is the optimal triangulation [16]. Delaunay methods were developed [17] that guaranteed the elements' quality, and significant effort placed on post mesh generation quality enhancement techniques [18].

To determine an approximate solution to a problem, a certain number of sampling points must be used. A problem whose solution changes rapidly requires more sampling points to achieve the same level of accuracy than that required for a solution that changes less rapidly. The lower graph in Figure 10 shows a solution that changes slowly, red line, and its approximation, solid line. This shows that for this problem a small number of sampling points can be used to approximate the solution to a high degree of accuracy, due to the smoothness of the solution. Conversely, the top graph in

Figure 10 shows a solution that changes rapidly. It is evident from this that the using the same number of sampling points as for the smooth solution, that the accuracy is greatly diminished. In order to achieve the same level of accuracy a larger number of sampling points would be required.

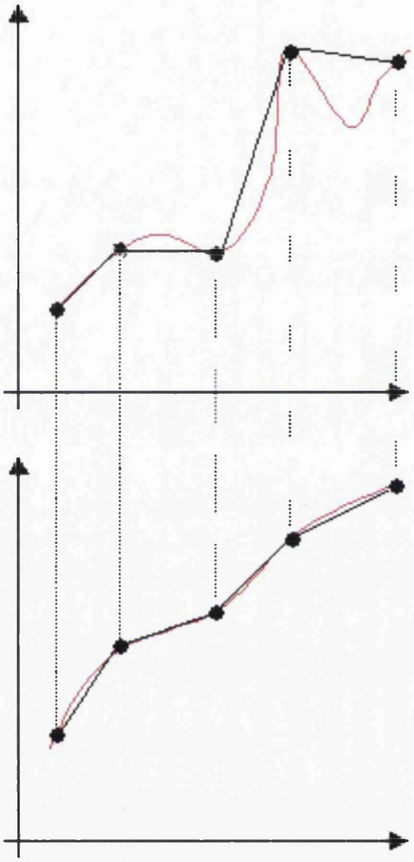


Figure 10 Solution resolution

High Reynolds number flows exhibit thin regions around any bodies present within the flow field, where the solution gradient is high. Within these regions, known as boundary layers, the flow exhibits highly directional properties where components of the flow solution, such as velocity, change rapidly in certain directions, and slower in other directions.

Due to the high solution gradients within the boundary layer a large number of points are required in order to resolve the solution accurately. However, since the solution is also directional with the boundary layer, then the point density need only be high in the direction of the rapidly changing solution. Analytical studies and experiments in wind tunnels have shown that the solution changes with distance from the solid wall more rapidly than along the wall.

It would be pragmatic then to use this knowledge to reduce the size of the mesh in certain directions, where the solution changes less rapidly, whilst keeping the dense point spacing required to resolve the solution where the solution gradient is high. Reducing the size of the mesh reduces the computational expense in terms memory usage, hard disk storage space and computing time.

Connecting a set of points that are not equally spaced can result in elements whose aspect ratio is high. Figure 11 shows a point spacing that has been created in an attempt to resolve a solution that changes rapidly along the Y axis, and more slowly along the X axis. Connecting these points results in stretched elements, where the longer edges are approximately aligned with the solution that is changing slower than that of the shorter edges.

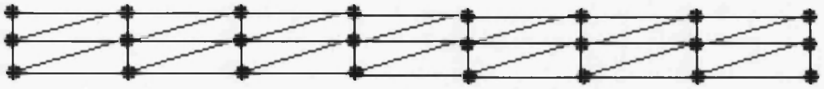


Figure 11 Directional point density leading to elements with large aspect ratio

Superimposing this mesh derived from a directional point spacing, upon that from an isotropic point density shows that the number of elements required to discretise the space is reduced for the directional case. It is clear then that using a directional point density to discretise a domain where the flow exhibits solution gradients that alter with direction can reduce the number of elements required to resolve the solution.

3.1 Literature Review

Research into the use of unstructured meshes containing anisotropic elements where the solution is highly directional began in the mid 1980s.

Initially, mesh adaptivity became a popular method to obtain directionally stretched meshes. Löhner and Morgan [19] and Löhner and Cebral [20] demonstrated a directional refinement method, which consists of determining error indicators for each edge within the mesh, and refining edges that exhibit errors greater than some threshold value. In classical refinement, all edges of an element are refined for a given error estimate, so that the resulting elements exhibit isotropic indicators, whereas here only the edges that exceed the error are refined resulting in stretched elements and higher point density in the direction of the solution. Mavriplis [21] published an extension of the Delaunay kernel, where the stretching and rotation of the elements is specified in given regions via a background mesh. Peraire et al [22] described remeshing the entire domain by means of adaptivity. Here a background mesh is used to control the three variables that control the mesh stretching, derived from the second derivative of the solution, the Hessian matrix. Initially the coarse mesh covering the domain contains two elements (the convex hull), and as the adaptation loop proceeds, described in Figure 31, the previous mesh.

In the mid 1990s, the methods for mesh adaptation using the extended Delaunay kernel, which is commonly used today to generate anisotropic unstructured meshes, emerged from Vallet [23], Borouchaki et al [24] and Castro-Díaz et al [25]. Using the Hessian matrix derived from a previous solution, the extended Delaunay kernel has been used for both remeshing and mesh adaptation. Tam et al [26], Borouchaki et al [27] [28] and Pain et al [29] demonstrated its use for both viscous and inviscid flow solutions. Castro-Díaz et al [30] demonstrated extensions and modifications to the metric definition to enable resolution at the boundary layers and multi-scale phenomena to be determined. Borouchaki et al [31] describe the use of the extended Delaunay kernel to generate adaptively triangular and quadrilateral meshes for flow simulations.

The method typically used to generate meshes upon surfaces described by geometric entities, such as NURBS, is to mesh the region in the parametric space. Generating elements in this space and then mapping back to the physical space provides regular mesh spacing on the curved surface. The mapping between parametric and physical space can require that elements generated in the parametric space are stretched. Since the extension of the Delaunay kernel allows the generation of anisotropic elements, the process can then be applied to surface meshing. Castro-Díaz and Hecht [32] and Yamada et al [33] discuss the use of the metric controlled Delaunay kernel for surface meshing. Lee [34][35] showed the use of metric specification to generate surface meshes containing triangles, which are later split into quadrilaterals, using an adaptive advancing front procedure.

3.2 Delaunay Triangulation

As discussed in 2.3, the Bowyer-Watson method of generating a Delaunay satisfying triangulation has been shown to be one to two order of magnitude faster in generating an equivalent sized mesh compared to other unstructured methods such as the advancing front method.

Dirichlet [36] described a method of dividing a domain described by a set of points into regions. These regions, known as Voronoi regions [37], are associated on a one-to-one basis with each point in the set. Each region exhibits the property that for each point P the associated region is the area that is closer to P than any other point in the set. The set of regions that fill the domain is known as a Dirichlet tessellation.

A Delaunay triangulation [38] of a set of points is derived from the Dirichlet tessellation. By introducing straight lines between points that share a Voronoi region boundary, a Delaunay satisfying triangulation of the set of points is generated. The Delaunay criterion is often referred to as the empty-circle criterion, and is essentially the evaluation of two distances; the distance from the centre point of the circle to that of the new point, Δ_2, and the radius of the circumcircle, Δ_1, (the distance between the centre point of the circle and one of the nodes of the triangle), Figure 12.

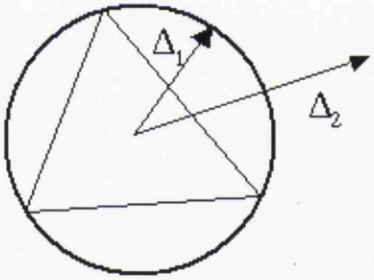


Figure 12 Distance for Delaunay evaluation

The vertices of the Dirichlet tessellation mark the point that is equidistant from each of the three points of a triangle[10]. For a triangulation to be Delaunay satisfying, each circle described by the three nodes of each triangle must not enclose any other point. Figure 13 shows the connection of four points using the Voronoi regions to indicate which points can be connected to produce a mesh that is Delaunay satisfying. The alternative connection, Figure 14, shows that the circumscribed circles of the elements are not empty.

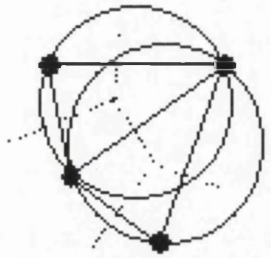


Figure 13 Delaunay satisfying connection

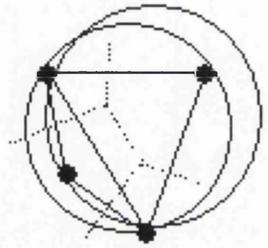


Figure 14 Delaunay violating connection

Efficient algorithms to generate a Delaunay triangulation from a predefined set of points were described by Green and Sibson [39], Bowyer[10] and Watson[11], in which a number of examples for the uses of these triangulations were given.

For numerical simulations the triangulation must be contained wholly within the boundary of the geometry. Thus, the algorithms were extended to accommodate this, by including boundary recovery [18] and automatic point insertion [7][18] routines. As described earlier, a minimum number of points are required in order to resolve the solution within the domain accurately. The actual number of points required is dependant on the level of accuracy required and the physics being modelled. To this extent, the number of points can be high in even the simplest simulation, such that the manual specification of the points is unfeasible. Automatic point insertion is used to make the Delaunay mesh generator capable of running without user intervention. The user places mesh control entities into the unmeshed domain in order to control the point spacing that will be produced by the automatic point insertion routines coupled into the mesh generator.

Mesh point density control entities, known as sources, as described in [18] consist of three variables; the position within the domain {x,y,z}, inner radius, outer radius and intensity or mesh spacing. The two radii describe the region in which the source operates at the coordinate given. Within the inner radius the intensity is constant, and within the region between the inner and outer radii the intensity decreases exponentially to the background intensity. Point and line sources are shown in Figure 15, for a two-dimensional geometry showing the inner and outer circles.

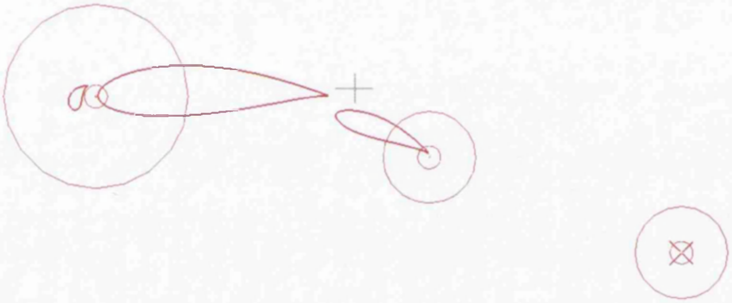


Figure 15 Point and line sources in two-dimensions

Boundary recovery routines are required to ensure that the triangulation is compatible with the geometry definition, described by edges (two dimensions) or triangles (three dimensions), which was used as the starting point for the process. The end product of a Delaunay triangulation process is a mesh that includes every point of the boundary and any interior points introduced by the point insertion routine. This does not, however, necessarily include the boundary edges or triangles that describe the problem. Boundary recovery is a post mesh generation process that enforces the boundaries of the initial problem into the final mesh. This process involves edge swapping in two-dimensions, and more complicated operations in three-dimensions. Numerical analyses of boundary recovery was discussed by Weatherill et al[40], and remains one of the challenges of Delaunay based unstructured mesh generation methods.

In order to construct a Delaunay satisfying triangulation, a coarse triangulation is defined, consisting of two elements. These elements cover the entire physical domain, and thus contain the boundary points, and are derived from the convex hull. Each boundary point is inserted into the triangulation one at a time, and the triangulation is updated to include the new point. The modification of the triangulation to include the new point is a purely local operation, and only affects the elements whose circumcircle includes this new point. The elements whose circumcircles include the new point are removed from the triangulation, leaving an empty space in the mesh known as a cavity. Using data structures to improve efficiency [10] [11], the time required to perform the search for elements whose circumcircles include the new point, and therefore construct the cavity, is kept to a minimum. The elements that are found to violate the Delaunay criterion are included into a cavity. This cavity is locally re-triangulated by finding the Dirichlet tessellation once more, for the new set of points. Figure 16 shows an example of a point inserted into a triangulation, with the elements that form the cavity shown and the circumcircles of the surrounding elements, with dashed lines. Evaluating the neighbours of the element that contains the point permits the construction of the cavity in an efficient manner. Once all boundary points have been inserted into the triangulation, the creation of new points within the domain can begin. Weatherill et al [18] describe a method of automatic point insertion within the triangulation. Point density is controlled by a combination of a background mesh and mesh control entities.

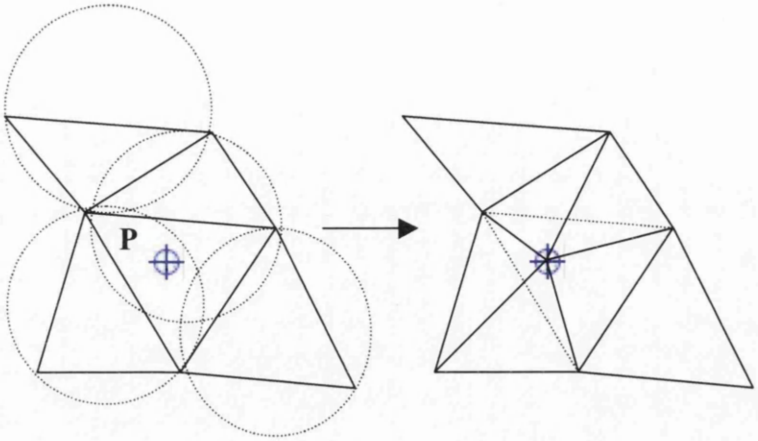


Figure 16 Isotropic Delaunay triangulation

Analysing a Delaunay mesh generator with the requirements set out in Chapter 1, it is noted that the list is fulfilled. Looking at only the technical aspects, as opposed to the implementation aspects such as ease of use, a Delaunay mesh generator is automatic, requiring no user input aside from setting the problem. The quality of the resulting mesh can be improved by post generation mesh cosmetic routines. The robustness of a Delaunay mesh generator is an implementation aspect, although the numerical aspects of the technique of generation (that of distance evaluation) are simple but can suffer from computation round off errors. The boundary recovery process an area that suffers significantly from numerical inaccuracies, although having been proved mathematically [40].

As explained previously, the use of stretched or anisotropic elements within a triangulation can reduce the computational expense of simulations where the solution is highly directional. The extension of Delaunay based methods to generate stretched elements is desirable due to the speed of these methods. In the anisotropic Delaunay method, the in-circle criterion is modified to allow ellipses to be used. The ellipses are defined for each node in a mesh, either from a pre-determined solution in the case for mesh adaptation, or by specification at points within the mesh. Figure 17 shows the insertion of a point P into a triangulation, where metrics have been specified for each point that describes ellipses at these points.

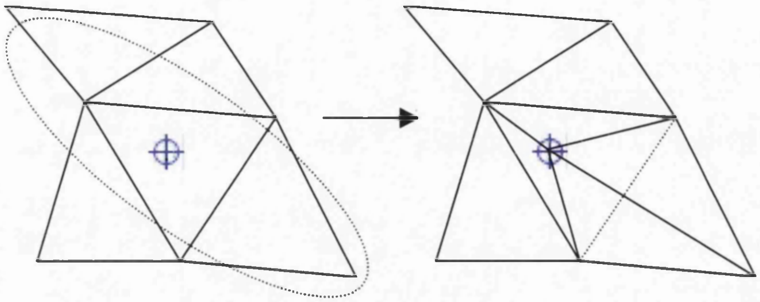


Figure 17 Anisotropic Delaunay triangulation

The metrics used to describe the ellipses used for triangulation describe a Riemannian space. An isotropic mesh is built within this space, which when transformed into the Euclidean space produces stretched elements. The metric M is a $d \times d$ matrix, where d is the dimension of the triangulation (i.e. two or three dimensions). In constructing a Delaunay triangulation, the distances comparison is performed repeatedly. It is these distance calculations that can be modified to provide the distance in the Riemannian space described by the metrics. The distance, $d(A,B)$, between two points A and B in Euclidean space is found using Equation 1.

$$d\left(A,B\right)=\sqrt{AB^{T}AB} \qquad \textbf{Equation 1}$$

Where the points A and B and the vector AB are defined as

$$A = \begin{Bmatrix} x_A \\ y_A \\ z_A \end{Bmatrix}, B = \begin{Bmatrix} x_B \\ y_B \\ z_B \end{Bmatrix}, AB = \begin{Bmatrix} x_B - x_A \\ y_B - y_A \\ z_B - z_A \end{Bmatrix} \qquad \textbf{Equation 2}$$

The distance between two points A and B in Euclidean space in the presence of a metric M is the modification of Equation 1 to include the metric M, Equation 3.

$$d_M(A,B) = \sqrt{AB^T MAB} \qquad \textbf{Equation 3}$$

Clearly, by using a metric that is symmetric and whose off-diagonal terms are zero the two distances are equal, $d_M(AB) = d(AB)$.

Given a suitable metric specification either monotonous across the computational domain or varying in space, George [41] suggested the use of a number of approximations in order to generate a triangulation that is Delaunay satisfying with respect to the metric map, which are described below.

As discussed previously, since the Delaunay criterion is essentially the comparison of two distances, the radius of the circumcircle, r, and the distance between the new node and the centre of the circle, $d(AB)$, the distances can be modified. Using Equation 3, the distance between the centre of the circle and the new point can be determined, in the stretched space defined by the metric. If this violates the Delaunay criterion, the circumellipse is not empty, then the element is added to the cavity. This criterion is described in mathematical terms in Equation 4.

$$d(AB) - r > 0 \qquad \textbf{Equation 4}$$

In order to determine whether the Delaunay criterion is satisfied in the case of anisotropic element generation, the position of point O$_k$ that is equidistant from the three points of the element in the presence of a metric is required. In two-dimensions, the solution of the system in Equation 5 will yield this position.

$$\begin{aligned} d_M(O_K, P_1) &= d_M(O_K, P_2) \\ d_M(O_K, P_1) &= d_M(O_K, P_3) \end{aligned}$$

Equation 5

In order to generate a mesh using the anisotropic metrics, the distance criterion containing the metric modification can be substituted for the existing Euclidean distance. In a simplified case where the metric is monotonic across the domain, a direct substitution is appropriate. However, if the metric varies across the domain, then a number of approximations suggested by George[41] are used in order to allow the mesh to conform to the metric.

The first approximation uses the metric at the point under consideration for insertion to determine the position of the point O$_k$, the centre of the ellipse, as the solution of a linear system, Equation 6.

$$\begin{aligned} d_{M(P)}(O_K, P_1) &= d_{M(P)}(O_K, P_2) \\ d_{M(P)}(O_K, P_1) &= d_{M(P)}(O_K, P_3) \end{aligned}$$

Equation 6

Hence the Delaunay measure is found by using the distance between the point P and O$_k$, in the presence of the metric associated with point P.

The second approximation for determining O$_k$ consists of extending the idea of approximation 1 by using the metric at the point under consideration for insertion *M*(P) and that of the element under consideration $M(\mathrm{P}_1)$, defined at the node that does not exist in the current cavity, shown in Figure 18. In this case the Delaunay criterion is modified to include the two distances; if the point is enclosed by both ellipses then the element is inserted into the cavity. Equation 7 shows the system of equations used to determine the corresponding centres of the ellipses, O_{k1} and O_{k2}.

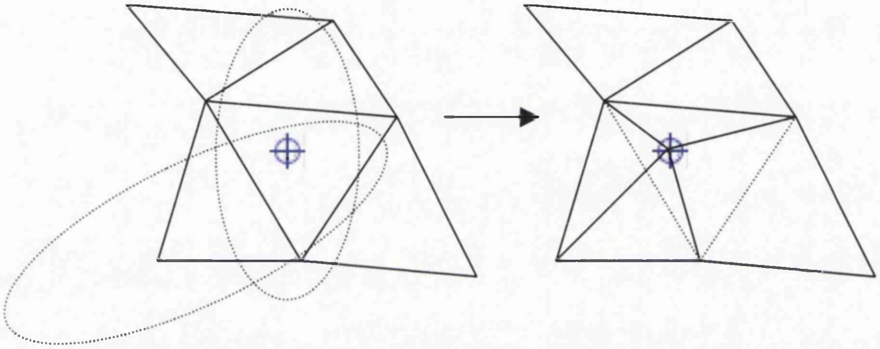


Figure 18 Approximation two diagram

$$\begin{aligned} d_{M(P)}(O_{K1},P_1) &= d_{M(P)}(O_{K1},P_2) \\ d_{M(P)}(O_{K1},P_1) &= d_{M(P)}(O_{K1},P_3) \\ d_{M(P1)}(O_{K2},P_1) &= d_{M(P1)}(O_{K2},P_2) \\ d_{M(P1)}(O_{K2},P_1) &= d_{M(P1)}(O_{K2},P_3) \end{aligned}$$

Equation 7

Extending this further to allow for the metrics defined at each node of the triangle with the node under consideration for insertion is described as approximation three. George remarks that whilst approximation one holds for two-dimensional meshing, it does not for three dimensions. The latter two approximations hold for two and three-dimensional work.

The Bowyer-Watson method for generating incrementally a Delaunay satisfying mesh has been extended to accept metrics that stretch the mesh in particular directions. The definition of suitable metrics across the computational domain must now be considered in order to allow the process to be applied. This can be done in one of two ways; the user definition of the metrics, using similar ideas to that for sources used to control the mesh point density, or *a-posteriori* methods for mesh adaptation, where the metric field is defined from an initial solution.

3.2.1 User specification of ellipses

Using knowledge of fluid flow and previous experience it is possible to define regions where anisotropic elements could be used to reduce the computational mesh size whilst still being able to resolve the flow properties to high levels of accuracy. As described earlier, mesh control entities are used in isotropic mesh generation to control the mesh point density across the computational domain, and it would be useful to extend these ideas to allow the specification of metrics across the computational domain.

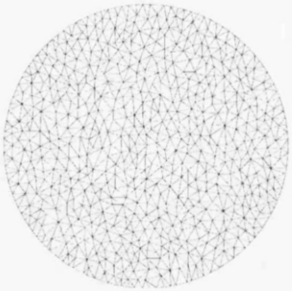
In order to allow the specification of simple variables that will control the mesh stretching in specific regions of the computational domain, it is necessary to understand the meaning of the metric, and what the metric represents.

A bivariate quadratic curve centred on the origin takes the equation shown in Equation 8. The coefficients in Equation 8 can be rewritten in matrix form, Equation 9. If $\det|J| > 0$, the determinant of J is positive, then Equation 8 defines an ellipse centred on the origin, where the variables *a*, *b* and *c* control the spacing in the principal directions along with the rotation from the principal axes.

$$ax^2 + 2bxy + cy^2 = 0 \qquad \textbf{Equation 8}$$

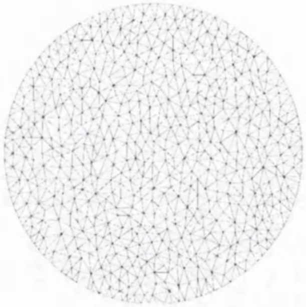
$$J = \begin{bmatrix} a & b \\ b & c \end{bmatrix} \qquad \textbf{Equation 9}$$

The aim then is to identify the coefficients *a*, *b* and *c* for an ellipse. Early tests with a simple circular domain show the effect of these coefficients, and from these it is possible to determine that the variables *a* and *c* control the stretching of the ellipse, and thus the stretching of the elements, and *b* the rotation of the ellipse. However, the coefficients do not directly indicate the stretching and orientation of an element created in the space defined by the metric J, since using $a = c, b = 0$ the mesh is isotropic for any *a*, shown in Figure 19.



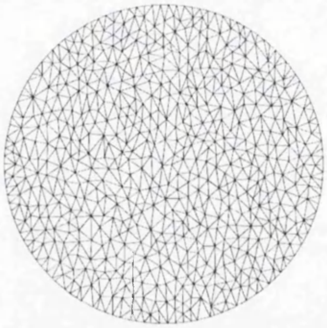
$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Figure 19 Mesh resulting from isotropic metric



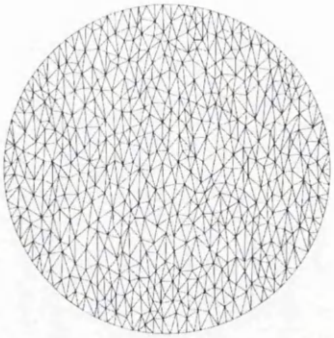
$$\begin{bmatrix} 1.2 & 0 \\ 0 & 1 \end{bmatrix}$$

Figure 20 Mesh resulting from anisotropic metric



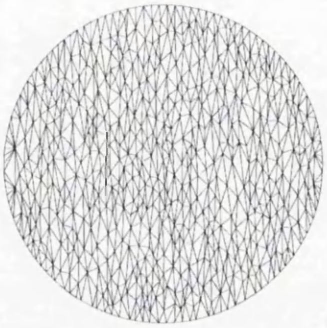
$$\begin{bmatrix} 1.9 & 0 \\ 0 & 1 \end{bmatrix}$$

Figure 21 Mesh resulting from anisotropic metric



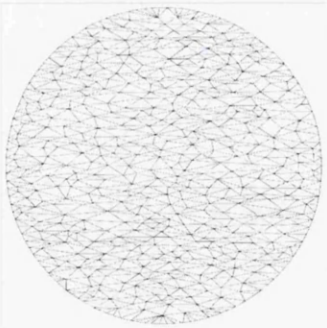
$$\begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}$$

Figure 22 Mesh resulting from anisotropic metric



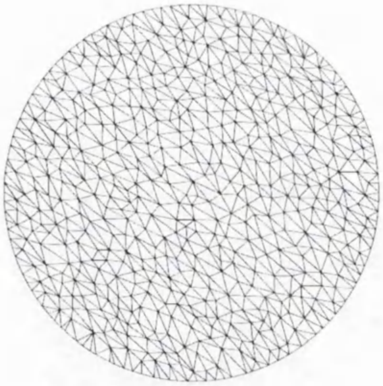
$$\begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}$$

Figure 23 Mesh resulting from anisotropic metric



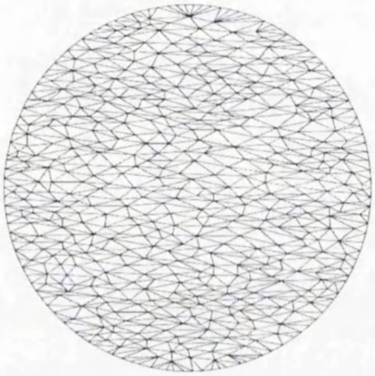
$$\begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix}$$

Figure 24 Mesh resulting from anisotropic metric



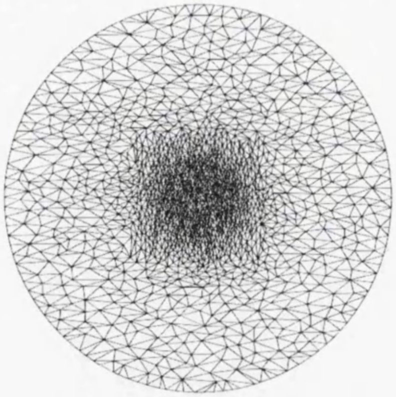
$$\begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

Figure 25 Mesh resulting from anisotropic metric



$$\begin{bmatrix} 1 & 0.25 \\ 0.25 & 10 \end{bmatrix}$$

Figure 26 Mesh resulting from anisotropic metric



$$\begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix} outer$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix} inner$$

Figure 27 Mesh resulting from anisotropic metric

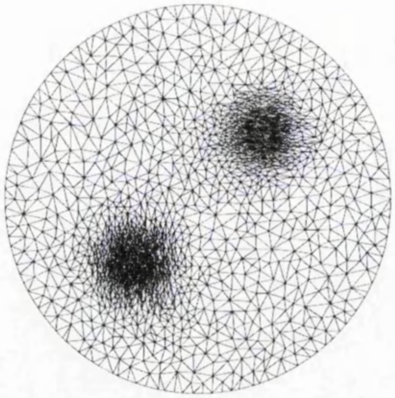


Figure 28 Point metric source specification without interpolation

Figure 21 to Figure 27 show the affect of varying a monotonous metric across a simple circular domain. From these examples it is clear that the first diagonal component, a, controls the spacing along the first principal direction, and the lower diagonal component, c, the stretching in the second principal direction, with the off-diagonal terms, b, controlling the rotation of the principal axes.

Given a desired stretching in the two principle directions and the angle of rotation of the ellipse from the principal axes it is possible to determine the coefficient b. Appendix A shows the substitution of the rotation, θ, to determine the off-diagonal terms as shown in Equation 10.

$$b = \frac{(c-a)}{2\cot 2\theta}$$

Equation 10

Using this definition for the rotation and George's [7] work in specifying the stretching, it is possible to place within the computational domain directional sources that control the stretching and orientation of the elements. Preliminary results, showing the use of line metric sources to control the stretching and orientation through a square domain are shown in Figure 29. Here line sources have been employed diagonally across the domain to control the point spacing. In these regions, line metrics have been specified to provide elements that are stretched and orientated at 45 degrees. The long edge lengths along the diagonals, and shorter edge lengths at normals to the diagonals show this. On the right of the figure, a close up of the centre of the domain is shown, where the two line metrics cross. It is at this point where metric intersection must be used, in order to determine the correct stretching of elements in this region. The figure shows that where the two metrics overlap, the intersection results in circles being defined which result in isotropic elements in the overlapping region. The metric intersection method is described in more detail in Section 3.2.3.

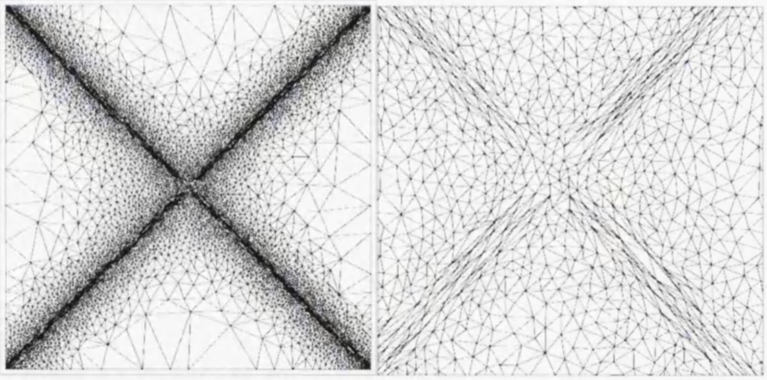


Figure 29 Line Metric specification for a simple square domain

3.2.2 Line and Point Metric Specification

The specification of metric to control the domain by the user consists of defining the principal radii of the ellipse, the rotation and the region over which the metric is employed

The point and line metrics are specified in the same manner as the source specification, although only a single radius is used to describe the area over which the stretching is constant.

3.2.3 Metric Intersection

Determining the controlling metric at a point where more than one metric has been specified requires that a form of averaging take place. The intersection of multiple metrics in order to retain the shape of one of the metrics is required for both mesh adaptation and mesh generation. For the case of mesh generation where point and line metrics have been specified by the user, and the circles defining the metric application area overlap, a single metric must be found to enable the extended Delaunay method to be used. Mesh adaptation can also produce multiple metrics throughout a mesh. The metric at each point of the current mesh is determined from an approximated solution, which can be derived for a number of variables such as Mach number or density (for a fluid dynamics simulation). Frey and George [8] suggested a number of methods to obtain a single metric at a point from any number of metric defined, using simultaneous matrix reduction. This method has been implemented for the mesh generation case, and results are shown in Figure 29, where the specification of line metrics diagonally across the domain interacts with larger point metrics. The interaction of these metrics can be seen on the right of the figure.

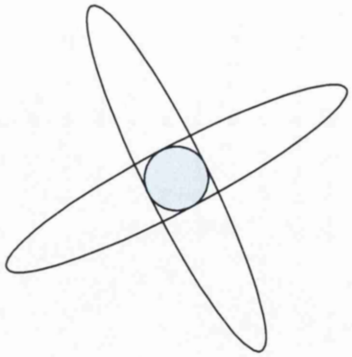


Figure 30 Metric Intersection example

3.3 Anisotropic Mesh Adaptation

Once the geometry has been created for a fluid flow simulation, knowledge of flow physics and the equation solution process is required in order to create a mesh with suitable point density to resolve the solution to required levels of accuracy. Since in some cases, error estimate analysis techniques can be applied to the final solution, it would be advantageous to use this knowledge to modify the initial mesh to minimise the error across the domain whilst striking a balance with computational cost. The use of error estimates and solution gradient at each point in a mesh with an initial solution can provide markers for refinement and coarsening of the mesh. The process of extracting these error estimates and modifying the mesh to reduce the error is known as mesh adaptation. The mesh adaptation and solution cycle is shown in Figure 31, an iterative automatic process until the error is reduced to required levels.

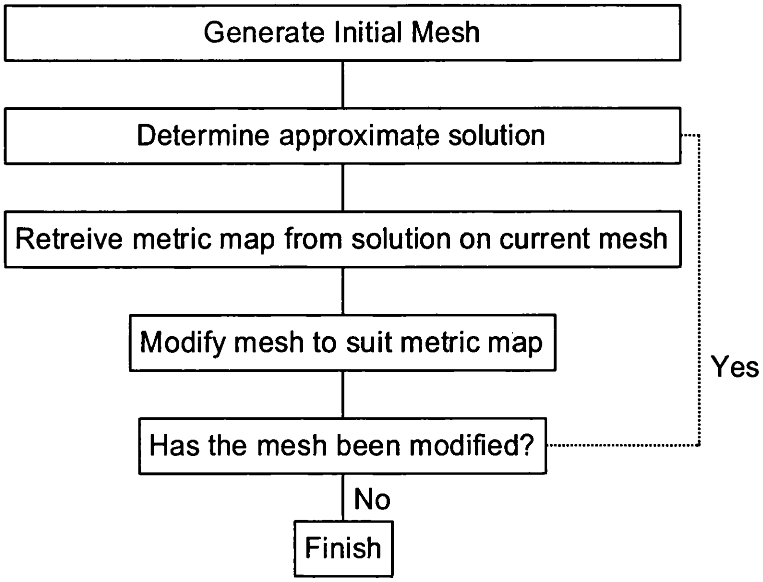

Generate Initial Mesh
Determine approximate solution
Retreive metric map from solution on current mesh
Modify mesh to suit metric map
Has the mesh been modified?
Yes
No
Finish

Figure 31 Mesh Adaptation Cycle

The anisotropic Delaunay method is a suitable candidate for use in mesh adaptation, since the metric map can be obtained from an initial solution, which is then used to control element shape and edge lengths within the mesh. The insertion of points into the mesh also allows a Bowyer-Watson type scheme to be used in conjunction with the metric map to update the mesh.

3.3.1 Metric Derivation

In order to derive the metric map across the domain used to control the mesh adaptation, the Hessian matrix of the solution, ψ, must be determined at each point in the mesh. The Hessian matrix is the matrix of second derivatives of a variable associated with the solution, shown in Equation 11 for a two dimensional solution. The scheme is based on that described by Pain et al [29].

$$H = \begin{bmatrix} q_{xx} & q_{xy} \\ q_{yx} & q_{yy} \end{bmatrix}$$ **Equation 11**

where

$$q_{xx} = \frac{\partial^2 \psi}{\partial x^2}, q_{yy} = \frac{\partial^2 \psi}{\partial y^2}, q_{xy} = q_{yx} = \frac{\partial^2 \psi}{\partial x \partial y}$$

The second derivatives are determined using a finite element method applied repeatedly. The first derivatives q_x and q_y are found on each element, e, initially using Equation 12.

$$\left.\frac{\partial \psi}{\partial x}\right|_e = \sum_j^3 \psi \frac{\partial N_j}{\partial x} \quad .$$
$$\left.\frac{\partial \psi}{\partial y}\right|_e = \sum_j^3 \psi \frac{\partial N_j}{\partial y}$$ **Equation 12**

The first derivatives on each node, P, are then found from the first derivatives on each element in the ball of P by Equation 13.

$$\left.\frac{\partial \psi}{\partial x}\right|_P = \sum N_j q_{x_j}$$
$$\left.\frac{\partial \psi}{\partial y}\right|_P = \sum N_j q_{y_j}$$ **Equation 13**

By substituting ψ for ψ' (the first derivative of the solution) into Equation 12, the second derivatives and components of the Hessian matrix are found.

The anisotropic Delaunay method is only valid for a metric that is symmetric and positive definite, and so the Hessian matrix must be modified into this form by using

Equation 14 suggested by [29]. R is the eigenvector matrix of the Hessian, and λ_i its eigenvalues.

$$M = R\begin{bmatrix} \tilde{\lambda}_1 & 0 \\ 0 & \tilde{\lambda}_2 \end{bmatrix} R^{-1}$$

Equation 14

where

$$\tilde{\lambda}_i = \min\left(\max\left(|\lambda_i|, \frac{1}{h_{\max}^2} \right), \frac{1}{h_{\min}^2} \right)$$

3.3.2 Dataset Derivation

A number of datasets are required to modify the mesh efficiently. The datasets consist of singly linked lists, which can be modified as the mesh structure changes throughout the mesh adaptation process, without resorting to rebuilding the whole dataset. The datasets required are:

- Elements connected to vertices
- Ball of vertices
- Edge structure of the mesh
- Neighbouring elements

3.3.3 Edge Evaluation

The mesh adaptation scheme is evaluated using the edges that form the elements. The scheme of adaptation is based on that by Borouchaki et al [27]. Each edge in the mesh is examined to determine the size of the edge in the space described by the metric $\hat{M}$ evaluated from the metrics defined at each node form the edge (M_1 and M_2) (Equation 15).

$$\hat{M}. = \frac{M_1 + M_2}{2}$$ **Equation 15**

The reference edge length in the current mesh is defined by the average edge length, l_{avg}, and the minimum and maximum edge lengths relative to this reference edge length are defined in Equation 16.

$$\begin{aligned} l_{ref} &= l_{avg} = \frac{\sum_{i=1}^{numedges} a_i}{numedges} \\ l_{\min} &= 0.8 l_{ref} \\ l_{\max} &= 1.2 l_{ref} \end{aligned}$$ **Equation 16**

The mesh adaptation loop consists of determining whether the edge length in the presence of the metric falls outside of the bounds set by l_{min} and l_{max}, using the criteria:

- If the edge length is less than l_{min} then a node is inserted along the edge
- If the edge length is greater than l_{max}, then a node is inserted at the mid-point of the edge, and the two end nodes are removed from the mesh

3.3.3.1 Interior Edge Refinement

The edges contained in the interior of the mesh, that is those edges that are not connected to a boundary node, are enriched by placing a point at the centre of the edge. The point must now be included in the triangulation. The metric for this point controls the cavity for the triangulation. Initially the two elements that form the edge that has been enriched are added to the cavity. Then, using the anisotropic Delaunay kernel, the neighbouring elements are inspected for inclusion.

A simple example of this is shown in Figure 32, where an edge has been enriched and the cavity enriched using adjacency.

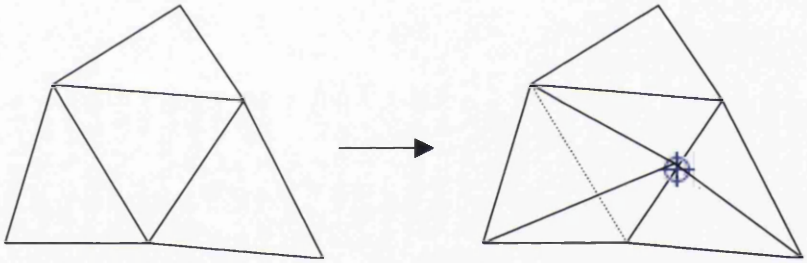


Figure 32 Interior Edge Refinement

3.3.3.2 Interior Edge Coarsening

Coarsening an interior edge consists of removing the edge and replacing it with a single node. The edge to be removed is bisected at its mid-point by a new node. The two nodes that formed the edge and the elements that were connected to these nodes are removed. The resulting cavity is re-triangulated using the metric at the new point, defined as the average of the two old nodes. An example of this is shown in Figure 33.

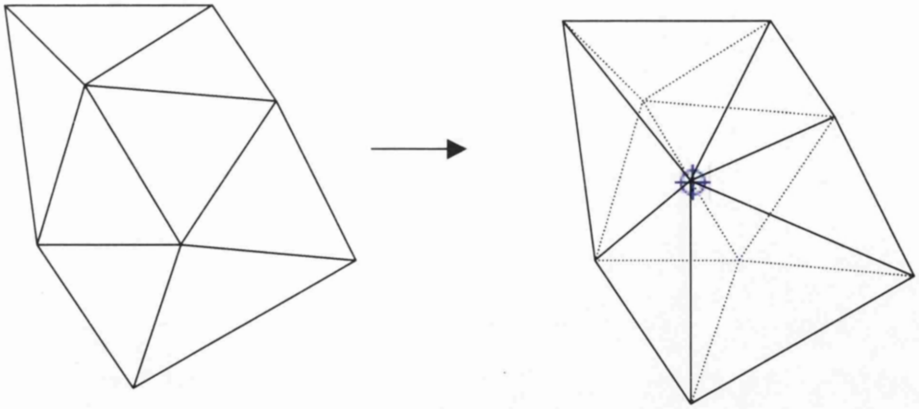


Figure 33 Interior Edge Coarsening

3.3.3.3 Boundary Edge Refinement

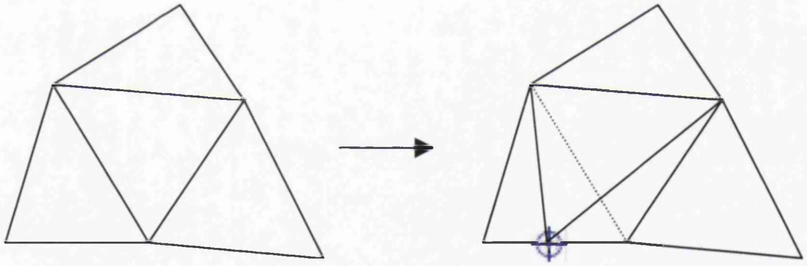


Figure 34 Boundary Edge Refinement

Boundary edge refinement represents a similar task to that of interior edge refinement. Figure 34 shows the refinement of a boundary edge by placing a node at the centre of the edge, and constructing the cavity. Care must be taken to ensure the edges introduced into the boundary are orientated correctly, and placed into the boundary edge list.

3.3.3.4 Boundary Edge Coarsening

The coarsening of a boundary edge is a similar process to that of an interior edge. An example of the removal of an edge that defines the boundary of a computational domain is shown in Figure 35. A single node at the centre of the old edge replaces the two nodes. This new node must be connected to the existing points either side of the removed edge, and all elements connected to the removed nodes are deleted and the resulting cavity re-triangulated. The node placement represents the new geometry, although it is not placed on the curve used to discretise the geometry

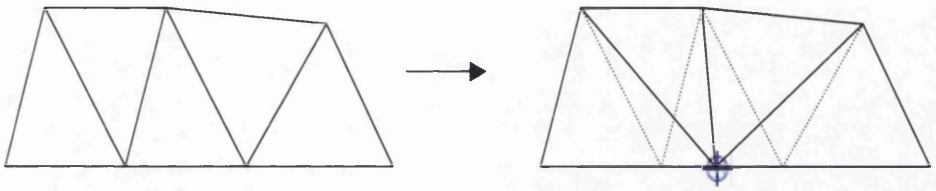


Figure 35 Boundary Edge Coarsening

3.3.4 Cavity Restriction

Adapting meshes using this scheme show that the ellipses, whilst controlled in size by using Equation 14, can enrich the cavity resulting in highly stretched elements. These elements show orientation in the correct direction, but do not have regular size. A modification to the cavity definition was developed, that restricts the cavity to neighbouring elements. In this way the ellipse is not empty, but the elements exhibit regular stretching, that results in a mesh where the neighbouring edge lengths vary in a regular manner. Figure 36 shows the a simulation of inviscid fluid flow across the NACA 0012 airfoil at an incidence angle of zero, with mesh adaptation without cavity correction applied, whereas Figure 37 shows the same simulation with cavity correction.

Adapted using Mach number, the non-modified cavity method (Figure 36) refines the region in front of the wing and three wake regions from the rear of the wing. The bow wave enrichment shows the highly stretched elements following the path of the wave, refining in the x-direction where the solution gradient is high. This results in highly stretched elements along the wave. The modified cavity (Figure 37) has picked out the same regions but has generated more regularly stretched elements in the bow wave regions.

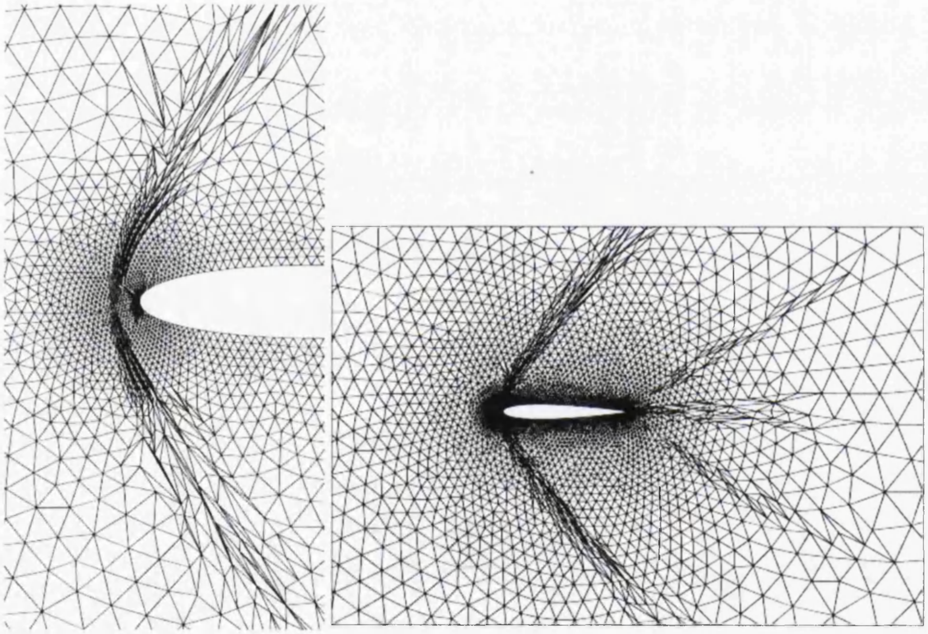


Figure 36 Non modified cavity

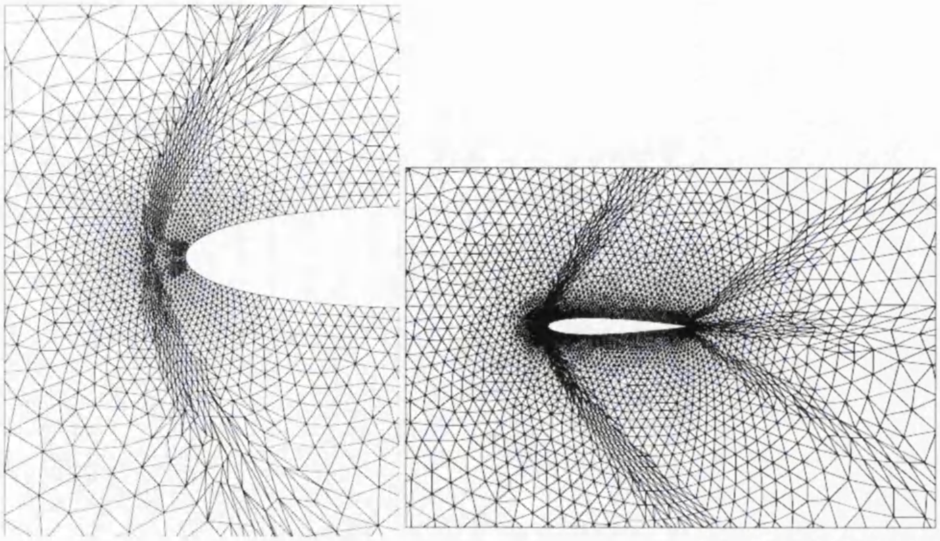


Figure 37 Modified Cavity

3.4 Results

3.4.1 Academic Example

In order to verify the metric calculation, a simple square domain was set-up, with a known solution. The unit square domain was meshed and a solution applied at the nodes in order for the metrics to be calculated. A circular region of radius 0.5, at the centre of the domain, was defined by a ring of width 0.1, where the solution is defined at each node i by:

$$\psi_i = 2$$

Inside this ring, the solution is ψ defined at each node i by:

$$\psi_i = \sin(x_i)$$

Outside the ring, the solution field is constant, defined by:

$$\psi_i = 1$$

The solution on the initial mesh is shown in Figure 38. The adaptation of this mesh would seek to refine the mesh in the direction of the solution, which within the central region is towards the right, as the sin wave solution varies with x coordinate. Hence it is expected that an increased point density is evident in the x direction, and a coarser density in the y direction.

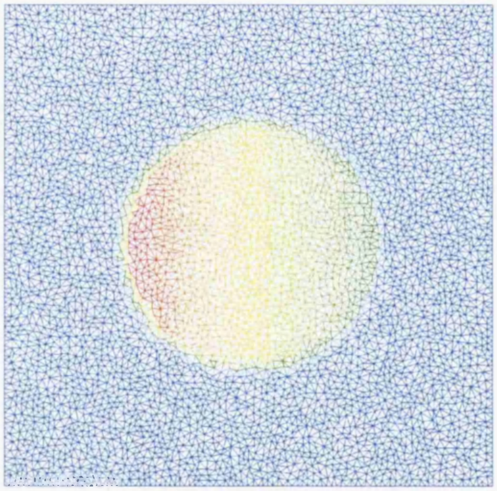


Figure 38 Initial mesh and solution

The mesh adapted with refinement only, that is without mesh coarsening and element removal is shown in Figure 39 with the new solution applied. The elements are stretched in the y direction, since in this direction the solution does not change, thus the edges that are orientated with the solution have been shortened in order to capture the sin wave more accurately.

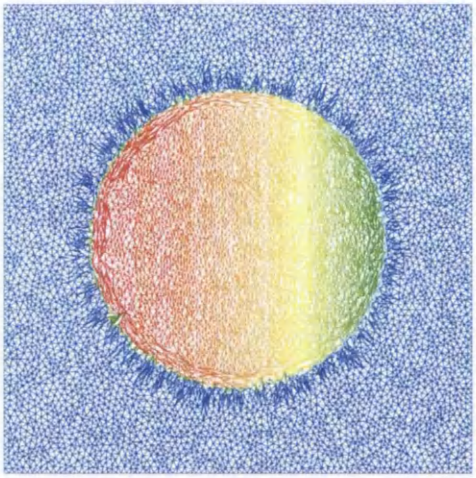


Figure 39 Adapted mesh and new solution computed from a previous mesh and solution

3.4.2 NACA Aerofoil Example

The NACA 0012 aerofoil has long been a standard geometry to determine the accuracy of flow solvers against experimental results. Here, the inviscid flow over the aerofoil has been used to adapt the initial mesh, in order to obtain accurate results within the shock region. The simulation was run with an incidence angle, α, of 2° and a Mach number of 0.7. A close-up of the geometry and initial mesh is shown in Figure 40.

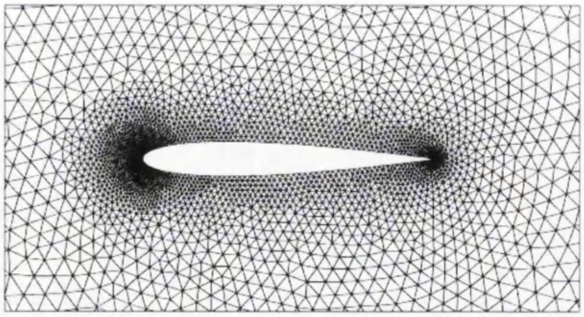


Figure 40 Initial isotropic NACA0012 mesh

Figure 41 shows a plot of Cp across the NACA aerofoil. The shock region is clearly evident and is the region in which the greatest variation between results is found. The adaptation here is isotropic, where each edge of a triangle is refined if a single edge of the triangle exhibits error exceeding a user-defined value. Three adaptation loops have been processed, each starting from the previous adapted mesh. As the edge length decreases within the shock region, the gradient of the results within the region increases. The original mesh exhibits smoothing of the shock at the beginning and end of the shock region, whereas the refined meshes at steps two and three resolve the steep gradient at the same points. Anisotropic mesh adaptation is shown in Figure 42. Evident in the graphs is an increase in solution resolution in the shock region, shown by a near vertical gradient for the first anisotropic adaptation loop. Subsequent adaptation loops exhibit reduced resolution quality, which can be put down to the element quality in this region. The solver used to determine the flow solution utilises a multigrid technique that enhances the convergence of the simulation. Figure 43 shows the coarsened meshes generated automatically to serve as the discretisation for the multigrid solver. Comparing these meshes with those shown in Figure 44, the coarsened meshes for an anisotropically adapted mesh, the stretched elements are forming a high proportion of the coarsened mesh. This mesh quality issue within the coarsened mesh affects the convergence of the simulation, and could be a reason for the solution accuracy diminishing as the adaptation cycles continue.

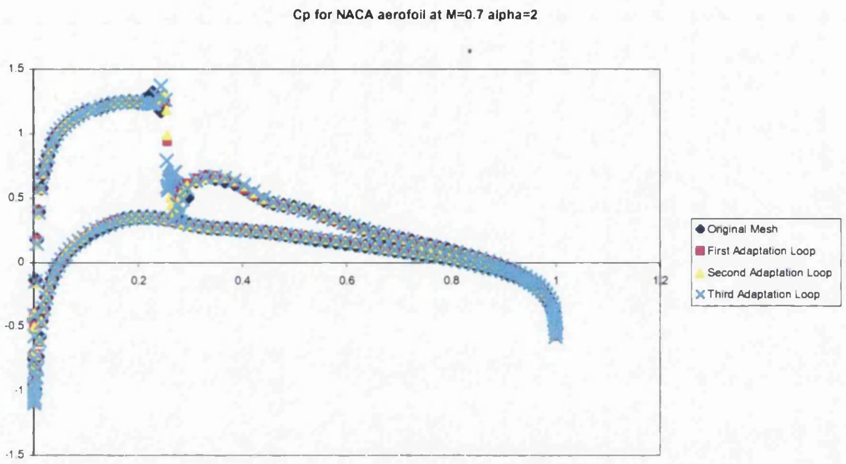


Figure 41 Cp plot for NACA Aerofoil with isotropic adaptation

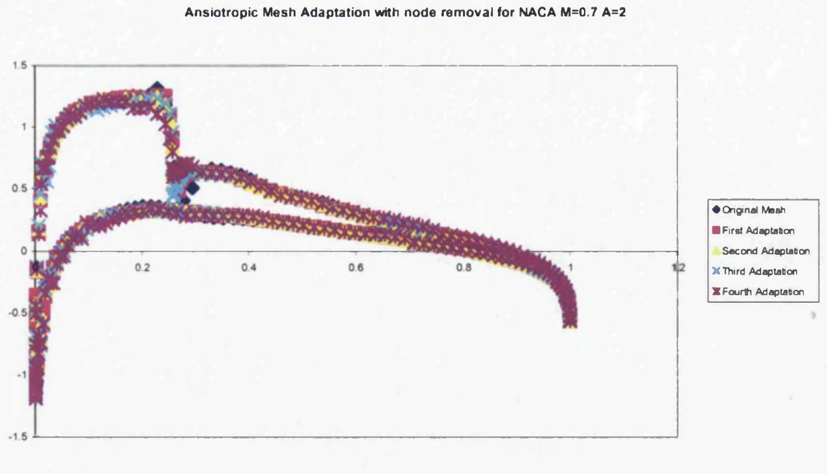


Figure 42 Cp plot for NACA aerofoil with anistropic adaptation

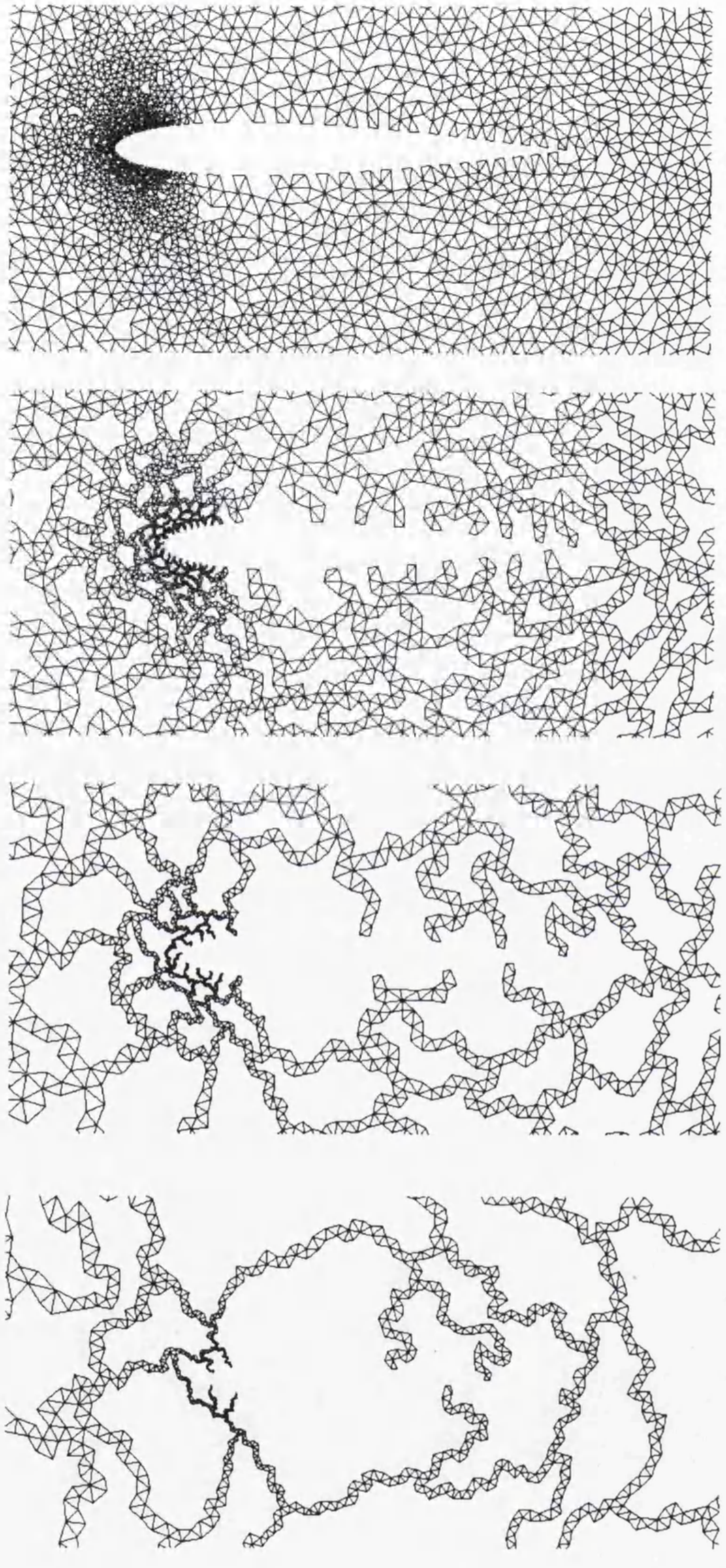


Figure 43 Isomesh adaptation multimesh schematic

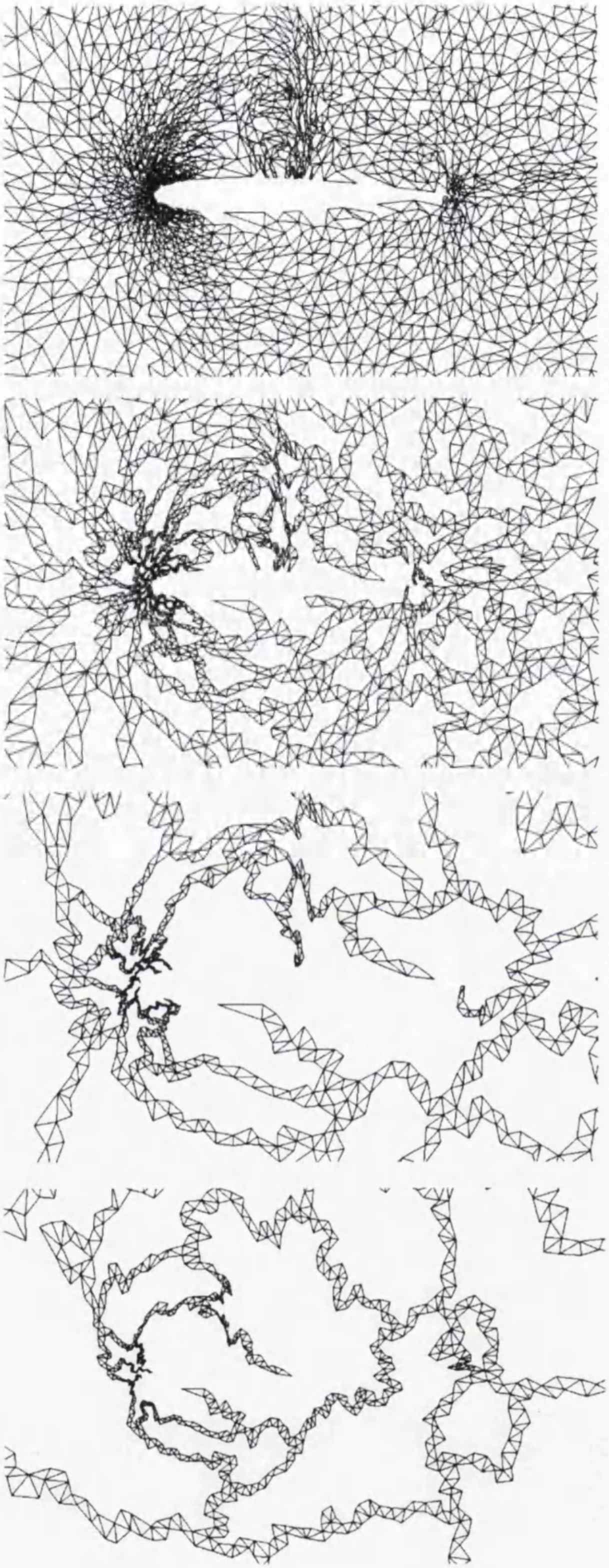


Figure 44 Anisoptropic adaptation multimesh schematic

3.5 Summary

This chapter has described the techniques that can be employed to generate stretched or anisotropic elements in order to reduce the size of the discretisation without detriment to the solution accuracy. The evolution of the anisotropic Delaunay method has been shown, and has been implemented in two-dimensions following the work of Borouchaki et al[27]. Work that remains uncovered, and would benefit from extension into three dimensions is the use of metric specified by the user using point and line metrics. These entities could be of great value in performing numerical simulations where the adaptation loop is not desirable. Here, using knowledge of flow physics, the mesh could be generated automatically to include stretched elements, in order to reduce the final mesh size. Issues with the multigrid methodology and stretched elements would benefit from further investigation and research in as much as it may be necessary to modify the mesh coarsening algorithms.

4 Parallel Mesh Generation

4.1 Introduction

The statement that computing power doubles every 18 to 24 months is well known [42]. The need for this power is ever-present within the computer simulation community, and methods to reduce simulation time and increase simulation complexity are researched. The concept of using more than one processor to increase the computing power is a logical step to make.

Parallel computing (using more than a single processor to complete a task) thus provides a means to increase the size, and hence the complexity, of a simulation whilst decreasing the time required to perform the simulation. Amdahl [43] recognised that the computation time can never exceed the time required for the serial sections of a parallel code. Indeed, many parallel programs exhibit levelling off of the speedup graphs (time versus number of processors) where the sequential algorithms dominate the computation. Gustafson [44] countered Amdahl's first law, by commenting that the purpose of parallel computing is to increase the size of problem that is possible to solve

[45]. For the computational simulation community this equals increased physics complexity modelling, requiring larger meshes to resolve the solution to the required level of accuracy

Many different aspects of their architecture can be used to classify computers. For parallel computing, these aspects are the number of instruction and data streams. A sequential computer containing a single processor and memory bank is known as a Single Instruction stream, Single Data stream computer (SISD). This follows the Von Neumann computer model. By increasing the number of instruction streams and/or data streams, parallel computers are obtained that exhibit distinct characteristics.

A Multiple Instruction stream, Single Data stream (MISD) model defines a computer more commonly referred to as a vector class computer; an example of a vector computer is the CRAY XMP. Here the parallelisation is performed, usually by the compiler, on an array level. For example to store a vertex in $\Re^3$ would require a two-dimensional array of length number of points and width three $\{x,y,z\}$. Any operations performed on the coordinates can therefore be distributed amongst three processors for computation.

Single Instruction stream, Multiple Data stream (SIMD) computers are more commonly known as Massively Parallel Processors (MPP). In this type of computer, there exists a control unit that distributes instructions to the processing elements (PE's), which in turn consist of an arithmetic unit and private memory. Each PE executes the same instruction concurrently, on different data held in the memory bank. Using these machines very high computational efficiency can be obtained for embarrassingly parallel problems. If a programming switch exists, such as an IF-THEN statement in Fortran, then each dataset must fulfil the same criterion. The occurrence of unmatched criterion results in wasted clock cycles whilst processors wait idle.

The last model of parallel computing is the Multiple Instruction stream, Multiple Data stream (MIMD). Arguably the most flexible of the parallel models, the computers consist of any number of processors (usually less than that of MPP's) with distributed or shared memory. The processors are typically cheaper than those of SIMD or MISD processors, and MIMD computers can be formed from networks of workstations.

Two main types of MIMD computers exist, providing a significant amount of computing power to the user, the differences in which lie in the memory distribution within the computer architecture. A distributed parallel computer can be thought of as geographically separate computers connected by a communication path, typically a network. The use of software such as CONDOR[46] or LSF[47] on individual workstations can increase the flexibility of distributed memory parallel computers. This type of software monitors the load on the available workstations and distributes jobs to the workstations that have little or no load. The software performs check-pointing of the programs if the workstation experiences a load other than that of the parallel process, and restarting the program on a different machine. Of the fastest computers in the world (listed at www.top500.org) over three-quarters are distributed memory supercomputers, such as the Cray T3 series and IBM SP series computers. These machines are similar to a network of workstations running load-levelling software, but connected by low-latency high bandwidth connections. Networks of workstations are typically connected via Ethernet or fast Ethernet, which exhibits high latency and low bandwidth performance.

A shared memory computer is formed by a number of processors each having access to a shared memory space. Computers such as the SGI Origin have distributed memory, but a method is employed via the operating system to allow the complete memory bank to be accessible from a single processor. The memory is connected by a fast connection capable of 1.6 gigabits per second, which hides a significant proportion of the latency involved in accessing memory that is physically separate from the processor.

This difference between the two types of parallel MIMD computers significantly affects the type of parallel program that is suitable to run on them, and the programming methods applicable. Writing a program for a shared memory machine is different to that for distributed memory computers. Since all processors have access to all the data for a program via the memory space, the use of message passing is not required; hence parallelisation of the algorithms within a program is better suited to a shared memory computer. A basic example of this would be to consider the multiplication of two square matrices, *A* and *B* of size $N \times N$, shown in Figure 45.

```
Do i = 1, N
  Do j = 1, N
    C(i,j) = 0
    Do k = 1, N
      C(i,j) = C(i,j) + A(i,k)*B(k,j)
    Enddo
  Enddo
Enddo
```

Figure 45 Sequential algorithm for matrix multiplication

For a shared memory computer this would be extended for parallel computation by using *N* processors each calculating the result for each row. This parallelisation can be performed due to the independent nature of the calculation. A distributed memory computer parallelisation of the algorithm would involve sending the data required for each row to each of *N* processors for computation, and then recompiling the final results in matrix *C*. For non-independent calculations, where the possibility of a process completing its section of the code before another and overwriting data still in use by other processes (such as iterative algorithms), use of barriers to synchronise the processes is required. These are placed into the code by the developer to force the program to wait at the points specified by the barriers until all processes have reached them.

The processing stage of a simulation represents a significant challenge in terms of computational efficiency. Complex physics simulations can require a mesh containing a large number of suitably shaped elements upon which calculations using complex algorithms are performed. Parallel computing has been used to speed up the time required to obtain the solution on these meshes containing many elements. Supercomputers such as the Cray T3E provide a large amount of processing power by using a large number of processors using memory that is exclusive to each processor; the memory available for each processor can range from 500Mb to 1Gb. A Delaunay sequential mesh generator typically uses approximately 100Mb per million elements generated, thus using a Cray T3E configured with 500Mb of memory per node it would be possible to generate approximately a five million element tetrahedral mesh. A supercomputer configured with 1024 processors would clearly be able to provide the solution to the simulation on a problem far larger than five million elements, and thus a parallel mesh generation method is sought. Shared memory machines, such as the SGI ORIGIN series computers, allow the generation of large meshes using sequential programs. This type of platform is suitable for large-scale simulations, since large amounts of memory are available to a single processor (in the region of 100 gigabytes of RAM), although limitations to the number of processes exist, and hence time penalties are incurred when compared to running on massively parallel processors. This would allow using sequential codes for mesh generation to generate meshes for simulations involving 250 million tetrahedra. In this case, the time taken to generate such a mesh would be prohibitively long. In the absence of a parallel mesh generator, other methods can be used to obtain partitioned meshes suitable for parallel solution. Partitioning libraries exist that can be used to give a good compromise between load balance and communication cost for mesh partitioning. For example, a mesh can be generated using traditional sequential algorithms, and then partitioned on the parallel machine that is to be used to determine the solution. Alternatively, large calculations have been performed using mesh enrichment to provide suitably large meshes. Here a coarse mesh is generated sequentially, and then partitioned and distributed to the parallel processors. For a simulation that uses constant mesh point density across the computational domain, the mesh enrichment process is highly scalable, and any load imbalance negligible [48]. This approach is not suitable for simulations such as CFD, where the mesh point density spacing varies across the domain.

In developing a parallel mesh generation method, a number of specifications should be adhered to:

- Reduce memory requirement to generate a given mesh, thus a processor should never hold more than one single sub-domain mesh at any one given time.
- Reduce generation time, all algorithms should be carefully designed without N^2 loops.
- Robustness – the process should be robust due to the large datasets that the process will be used to generate

The granularity of the parallel scheme employed is also important, and could have significant impact on the performance of the program. The granularity of a parallel algorithm describes the computation to communication ratio. The solution stage of the simulation process in parallel form is typically a fine-grained parallel algorithm, for explicit type calculations. For each time step, the boundary nodes are computed first and then communicated to the sharing sub-domains. Once the communication has been performed, the solutions at the interior nodes are determined. Since this is performed at each time step, the communication cost is high, and minimising this by careful domain decomposition can reap rewards in terms of wall clock time. The scheme chosen for the parallel mesh generator is coarse grained, this will allow the generator to be instrumented on physically separate machines using high latency, low bandwidth inter-process communication devices, typically ethernet. The sub-domains are written to disk, and a small message is sent to the corresponding slave to read the sub-domain. For either domain decomposition or volume mesh generation the computation involved for each message is large, and thus the generator represents a coarse grained algorithm.

4.2 Literature Review

Parallel mesh generation became an important field of research in the early 1990's when parallel computers became increasingly common. Classified into three groups in [49], parallel mesh generators take one of three forms, concerning the handling of the interface regions between processors. The classifications are:

- Parallel generators that mesh the inter-domain regions prior to volume meshing

- Parallel generators that mesh the inter-domain regions as the volume meshing is performed
- Parallel generators that mesh the inter-domain regions after the volume meshing

Initially in two dimensions, methods to perform parallel mesh generation were investigated. Löhner et al [50] described the parallelisation of the advancing front procedure using Cartesian, quadtree/octree or background mesh domain decomposition. In this work, the inter-domain regions are discretised after the sub-domains then the final mesh is assembled. A parallel Laplacian smoother is applied to the sub-domains with element migration across the inter-domain boundaries to improve the quality of the final mesh. Verhoeven et al [51] demonstrated parallel Delaunay mesh generation using domain decomposition. Initially a coarse mesh is built by connecting the boundary points without inserting points into the domain, and RSB [52] decomposition applied. The boundaries of the domain decomposition result are discretised with the correct point spacing, and each individual sub-domain meshed using a constrained sequential Delaunay mesh generator. Lämmer et al [53] has described the use of domain decomposition using inertial partitioning on the CAD geometry for solid mechanics simulations. Introducing lines to partition the geometry into separate sub-domains and applying sequential Delaunay algorithms on the resulting boundaries. Topping et al [54] have parallelised the advancing front procedure to generate quadrilateral meshes, using a processor farming approach. By employing manual domain decomposition, performed by the user, the meshing is performed by adaptively re-meshing each sub-domain with quadrilateral elements.

In three dimensions, the methods developed in two dimensions have been successfully extended. Löhner [55] has shown the extension of the parallel advancing front scheme into three dimensions to generate meshes containing up to 100 million tetrahedral elements. Performing diagonal swapping, bad element removal and Laplacian smoothing in parallel ensures element quality without loss of computational efficiency. Coupez et al [56] have shown the application of parallel re-meshing for solid mechanics problems although report significant mesh quality problems due to constraints applied to extract high computational efficiency. Parallel meshing is described in two-dimensions consisting of partitioning a non boundary-conforming mesh and meshing the resulting sub-domains in parallel. Rypl et al [57] have demonstrated a similar approach to Lämmer et al [53] of partitioning the CAD geometry in three dimensions, although have reported problems with model topology that have incurred restrictions. Said et al [58] demonstrated the extension of the work by Verhoeven et al [51] into three dimensions using coarse meshing and partitioning, and in [59] demonstrated the geometrical partitioning method extended within this thesis.

Parallelising the Bowyer-Watson algorithm has been accomplished by a number of research groups [60][61]. The process consists of inserting points in parallel into the mesh that is distributed across a number of processors. Calculating the cavity and creating the tetrahedra is highly parallel if the cavity is contained on a single processor. However if the cavity extends across a number of sub-domains, the elements that exist whose circumspheres include the new point must be requested and communicated across the partitions. Chrisochoides et al [62] have made a significant contribution to the parallelisation of the Bowyer –Watson algorithm, by reducing the latency incurred by the request and receive procedures required to form the cavity when the circumsphere encroaches across sub-domain boundaries, a speed up factor of 6 has been reported for generating one million tetrahedra. de Cougny et al [63] have demonstrated the parallelisation of the advancing front method using a distributed octree. This tree is partitioned in parallel using recursive bisection, which are then meshed in parallel. The inter-domain regions are left empty, and meshed once the sub-domain meshing stage has completed.

Wu et al [64] demonstrated a parallel adaptive meshing technique, involving coarse initial meshing and domain decomposition to provide the sub-domains upon which an adaptive meshing scheme is performed to retrieve the final fine meshes required for simulation.

4.3 Isotropic Parallel Mesh Generation

4.3.1 Domain Decomposition Technique

A domain decomposition step is required before the generation of volume elements can begin, in order to reduce the single volume mesh generation tasks to a set of smaller sub-tasks. This is to sub-divide the domain into smaller sub-domains, which can then be farmed amongst the available processors.

The domain decomposition procedure must fulfil a number of criteria to be suitable for use as a mesh generation tool. The procedure should be robust, in that any errors that could occur, such as invalid surface mesh definition, can be handled within the program. The procedure should be restorable; if the machine that the program is operating on fails; the user should be able to restart the code with a different number of processors and/or sub-domains. This capability provides a high degree of flexibility, in that the program can be stopped at any point during the domain decomposition procedure and then restarted on a different computer all together.

The decomposition procedure should also exploit any avenues for parallelisation available in order to minimise idle processor time. The principle aim of the domain decomposition tool, however, is to sub-divide the mesh generation problem into smaller sub-problems, each capable of running across multiple processors, utilising the memory available. The aim of the parallel mesh generation strategy as described in section 4.1, is to minimise the memory requirement for generating large meshes and to reduce the mesh generation time as a by-product of parallelisation. Thus, dividing the problem into smaller independent sub-problems, allows the use of distributed machines containing limited amounts of memory.

A domain decomposition scheme such as this relying on a surface mesh, and planar cuts to separate the sub-domains, can give rise to unbalanced workload for parallel simulation solvers, if the volume mesh sub-domains produced by the parallel mesh generator are used as the partitioning for the solution step. The domain decomposition step applied within the parallel mesh generator is not intended to produce volume mesh sub-domains that are suitable for simulations where the mesh point density is not constant throughout the computational domain (such as CFD simulations). The final mesh of the whole domain is brought back together in a manner that minimises any memory usage, and is described in more detail in section 4.3.4.2.2. Once the mesh has been brought together, public domain mesh partitioning codes, typically graph based, as described in section 4.3.2.1, are used to partition the volume mesh for sub-domains, that minimise any communication, and minimise the variance in element / point count in the mesh sub-domains.

The load balancing that can be applied to the domain decomposition scheme involves balancing the number of faces as far as possible between the sub-domains. The algorithm sub-divides only those sub-domains that have a boundary face count higher than that of a cut-off value, α, an optimal figure of which has been found to lie within the region:

$$\frac{Nsub}{8} \geq \alpha \geq \frac{Nsub}{16}$$

where *Nsub* is the number of sub-domains as set by the user. Using this cut-off value allows the domain decomposition to bisect the sub-domains that have a higher number of boundary faces, and possibly a high workload for volume mesh generation. This method then exhibits octree type decomposition, as shown in Figure 46, a generic wing-body geometry. The generated inter-domain meshes are shown in blue, whilst the geometry is yellow. The octree type decomposition, where regions of high point density have been decomposed can be seen around the wing region.

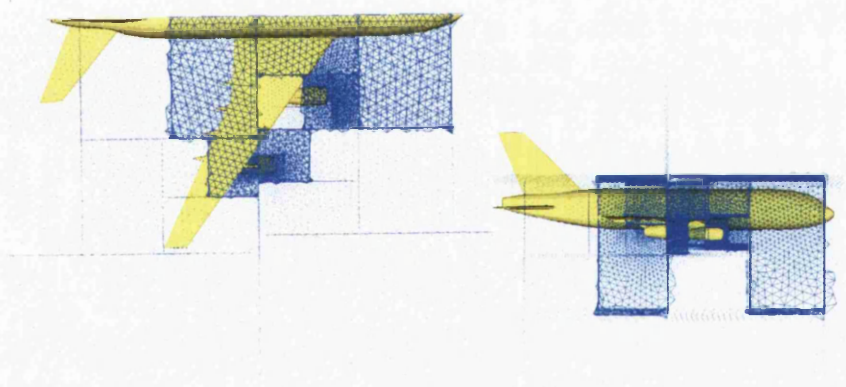


Figure 46 Generic wing-body geometry showing domain decomposition and inter-domain meshes in blue

Clearly, the number of sub-domains as set by the user may not allow the sub-domains to become coupled closely enough to allow the number of boundary faces per sub-domain to become balanced. A method to balance the domains to ensure the workload of the volume element generation step is balanced as far as possible, to utilise the parallel architecture fully, was devised and implemented, and is described in section 4.3.2.2.

Bisecting a surface mesh with a plane allows the domain described by the surface mesh to be separated into two halves. A mesh generated on this plane should accurately represent a cut through a mesh generated on the whole surface of that domain. In this way, the two smaller sub-domains can be generated independently, and mesh quality in the region of the plane should not suffer.

The domain decomposition process is written in such a way that once the first bisection has been completed and validated (section 4.3.1.10) then multiple processors can be used. The sub-division of a surface mesh is an independent process, and hence any number of sub-domains can be sub-divided at any given time provided the number of processors is available. Here, dynamic load balancing is used (section 4.3.1.3) to cope with an imbalance in processors and sub-domains.

The iterative process is shown more clearly in Figure 47. Starting from the top, the surface mesh of triangles describes the computational domain. A plane is introduced into the domain orthogonal to the longest axis, in this case the *x*-axis. Extracting the nodes that intersect this plane, and mapping into two-dimensions, allows the two-dimensional mesh generation process to take place. This mesh "fits" into the surface mesh, in as much as the point density on this plane conforms to that described by mesh control entities. This now separates the original surface into two independent sub-domains, which can now also be sub-divided.

The cutting plane can either be placed orthogonal to an axis (*x, y* or *z*) or the plane can be placed at an angle to an axis, described in Section 4.3.1.4.

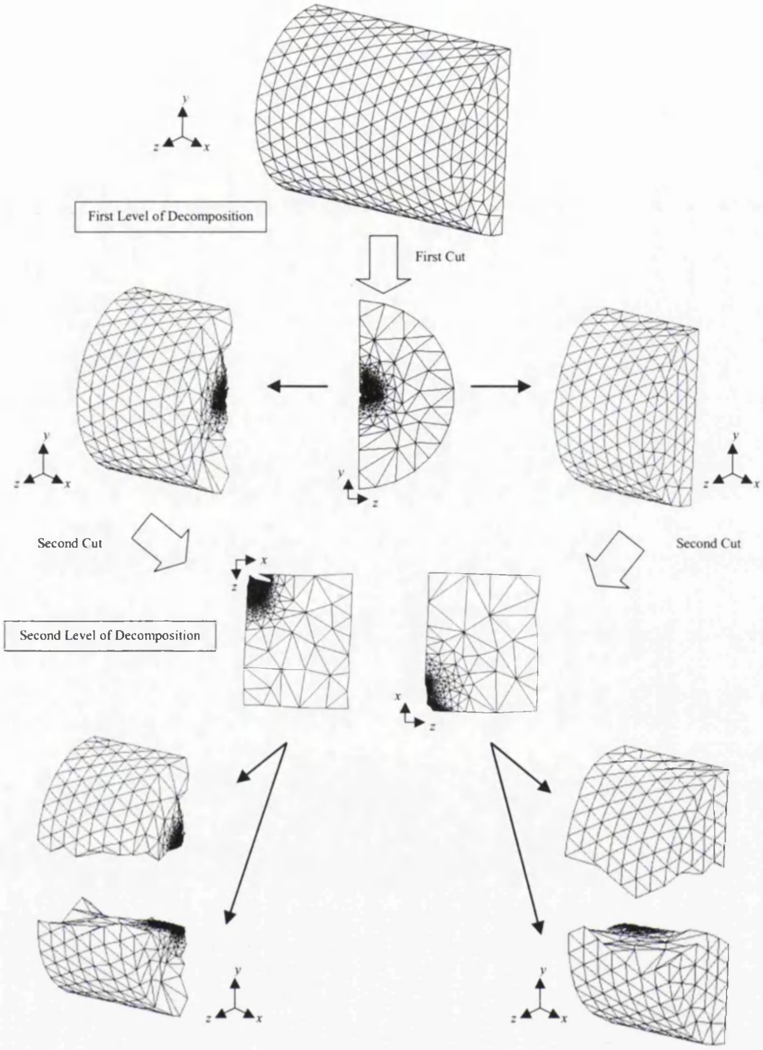


Figure 47 Domain decomposition strategy

The steps required to intersect a plane with the surface mesh are shown in Figure 48. The master process handles the problems (sub-domain boundaries) to be sub-divided, and send the messages to the slaves.

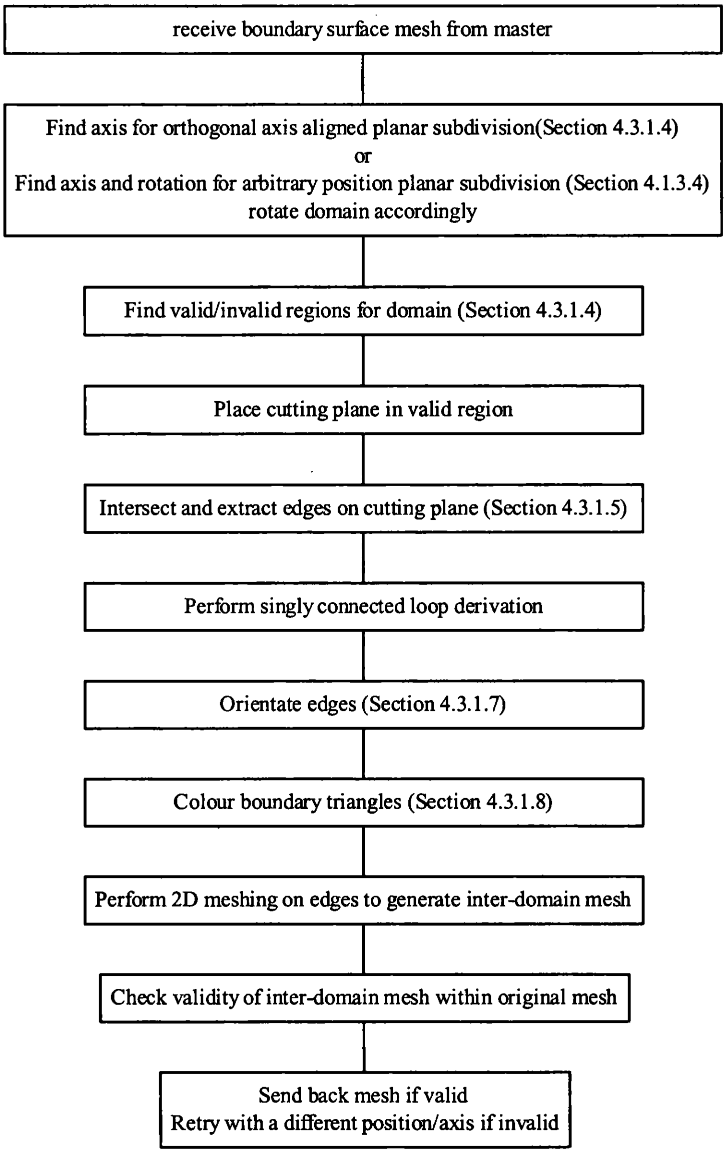

receive boundary surface mesh from master
Find axis for orthogonal axis aligned planar subdivision(Section 4.3.1.4)
or
Find axis and rotation for arbitrary position planar subdivision (Section 4.1.3.4)
rotate domain accordingly
Find valid/invalid regions for domain (Section 4.3.1.4)
Place cutting plane in valid region
Intersect and extract edges on cutting plane (Section 4.3.1.5)
Perform singly connected loop derivation
Orientate edges (Section 4.3.1.7)
Colour boundary triangles (Section 4.3.1.8)
Perform 2D meshing on edges to generate inter-domain mesh
Check validity of inter-domain mesh within original mesh
Send back mesh if valid
Retry with a different position/axis if invalid

Figure 48 Flowchart for slave process operations for domain decomposition

4.3.1.1 Master-Worker Processor Structure

The processor structure of the parallel mesh generator takes the form of a master-worker strategy. The master process deals with handling and sorting tasks, which are then processed and distributed amongst the available worker processes which perform the required tasks, including mesh subdivision, volume mesh generation and mesh cosmetics. The worker processes are independent from each other, and so no communication is required between them.

4.3.1.2 Message Passing Libraries

In order to develop a parallel mesh generator two options exist; explicit message passing for distributed computing and compiler parallelisation. Explicit message passing utilises message passing libraries such as MPI [65] and PVM [66]. These libraries provide a means by which it is possible to start parallel programs with an arbitrary number of processes, running on any number of computers connected by some form of network (Ethernet, Miranet etc). Using the MPI library for portability, the parallel mesh generator can operate on either distributed or shared memory parallel platforms.

4.3.1.3 Dynamic Load Balancing

In order to cope with an imbalance of processors and sub-problems, dynamic load balancing is used [67]. Consider the case for parallel mesh generation where the surface mesh has been decomposed into 8 sub-domains. If only four processors are available the dynamic load balancing provides a means by which the eight sub-domains can be meshes in parallel. The first four sub-domains are sent to the available processors, which then generate the volume meshes in these sub-domains. Once a mesh has completed, a message is sent back to the waiting master, who then builds the next sub-domain and sends to the processor that has just completed. In this way, any imbalance in the workload for each processor is balanced as far as possible and an imbalance between processes and tasks is also balanced.

4.3.1.4 Planar Placement

The planar cut can either be placed orthogonal to the longest axis of the sub-domain, or rotated about an axis according to an analysis of the inertia matrix of the sub-domain. Farhat and Lesoinne [68] suggested the use of inertia algorithms to partition finite element meshes for parallel finite element solution, more recently Lammer and Burghardt [53] show the application of inertia algorithms to generate triangular and quadrilateral meshes. By extending the work in [53][68] into three dimensions, the inertia method has been applied in order to determine the axis and principal direction (and hence the angle of rotation,θ) to place the planar cut.

The choice of axis comes from the eigenvalues for the inertia matrix, and the angle of this plane from the eigenvectors. The inertia matrix is written as shown in Equation 17.

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}$$

$$I_{xx} = \sum_{i=1}^{N} (y_i - y_{cg})^2 + (z_i - z_{cg})^2$$

$$I_{yy} = \sum_{i=1}^{N} (x_i - x_{cg})^2 + (z_i - z_{cg})^2$$

$$I_{zz} = \sum_{i=1}^{N} (x_i - x_{cg})^2 + (y_i - y_{cg})^2 \qquad \textbf{Equation 17}$$

$$I_{xy} = I_{yx} = \sum_{i=1}^{N} (x_i - x_{cg})(y_i - y_{cg})$$

$$I_{xz} = I_{zx} = \sum_{i=1}^{N} (x_i - x_{cg})(z_i - z_{cg})$$

$$I_{yz} = I_{zy} = \sum_{i=1}^{N} (y_i - y_{cg})(z_i - z_{cg})$$

where cg denotes the centre of gravity of the domain

Taking the eigenvalues of the matrix *I*, *E_i*, and the eigenvectors *E_v*, the largest *E_i* is used as the axis for subdivision, and the angle between this axis and the corresponding eigenvalue as the angle of rotation. Since this type of decomposition is simply a rotated plane, it is possible to rotate the sub-domain in the opposite direction to obtain an inclined cutting plane through the sub-domain. The domain must be rotated about the origin, centred on the centre of gravity of the domain.

The rotation matrix A has three forms, dependant on the axis that the domain is being rotated around. For x-axis rotation the matrix is of form A_1, for y-axis rotation the matrix takes the form A_2 and for z-axis takes the form A_3 (=A_1), (Equation 18).

$$A_1 = A_3 = \begin{Bmatrix} \cos\vartheta & 0 & \sin\vartheta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\vartheta & 0 & \cos\vartheta & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

$$A_2 = \begin{Bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\vartheta & -\sin\vartheta & 0 \\ 0 & \sin\vartheta & \cos\vartheta & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

Equation 18

The domain must be rotated centred around the origin, and hence two matrices are formed that allow the domain to be moved to the centre and away again after rotation, these are shown in Equation 19.

$$B = \begin{Bmatrix} 1 & 0 & 0 & -x_{cg} \\ 0 & 1 & 0 & -y_{cg} \\ 0 & 0 & 1 & -z_{cg} \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

$$C = \begin{Bmatrix} 1 & 0 & 0 & x_{cg} \\ 0 & 1 & 0 & y_{cg} \\ 0 & 0 & 1 & z_{cg} \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

Equation 19

The rotation of the domain can now be written as a transformation to the origin, a rotation and a transformation back to the original position, Equation 20. Where X_{old} is the vector of coordinates for each point within the mesh, with a fourth component equal to one. The fourth component in the vector, and the fourth row in each matrix *A,B,C* is a check that should be equal to one in the result vector X_{new}. This check is inserted to ensure that the transformation has been completed successfully.

$$X_{new} = ((A \times B) \times C) \times X_{old}$$ **Equation 20**

Preliminary results, showing the decomposition of the computational domain of the EADS Gulfstream into 16 sub-domains is shown in Figure 49. A close-up with the aircraft and cutting planes coloured by sub-domain number is shown in Figure 50.

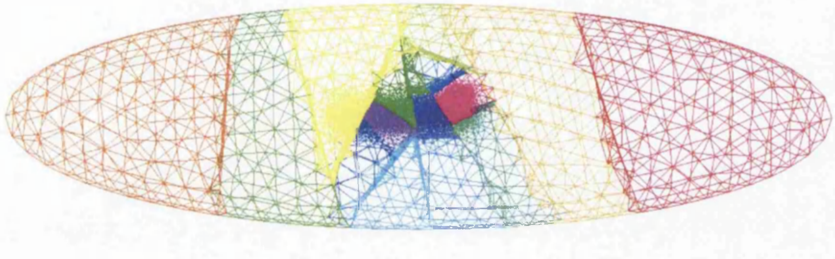


Figure 49 Arbitrary cutting plane decomposition of the EADS Gulfstream

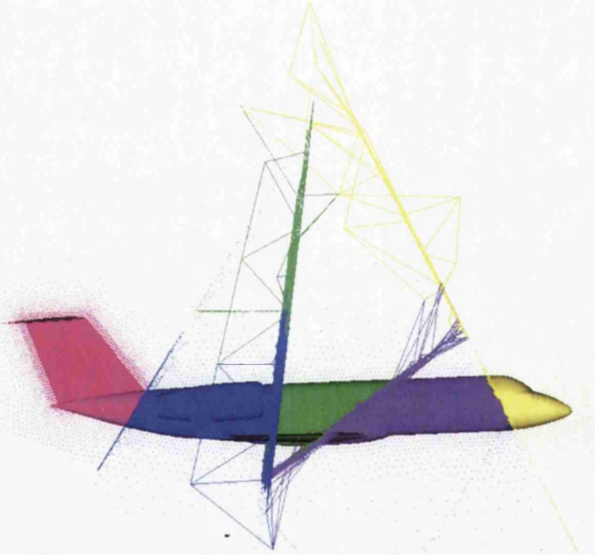


Figure 50 Close-up of domain decomposition

The position of the planar cut that forms the inter domain boundary is critical for mesh quality and successful domain decomposition. If the plane is placed too close to natural boundaries, such as leading and trailing edges of wing tips, for a specified edge length / point density, then the elements generated in the final volume mesh can be of poor quality. As an example, Figure 51 shows a two-dimensional case of a wing tip and cutting plane (dashed line). A simplified case, the shortest distance between the leading edge and cutting plane, δ, is less than the specified edge length, h, within the region, thus giving rise to elements that do not correspond to the mesh density specified by the user. Post processing steps have been taken to alleviate this occurrence, in three dimensions, where the mesh quality procedures described in 4.3.4.1 are unable to remove bad shaped elements.

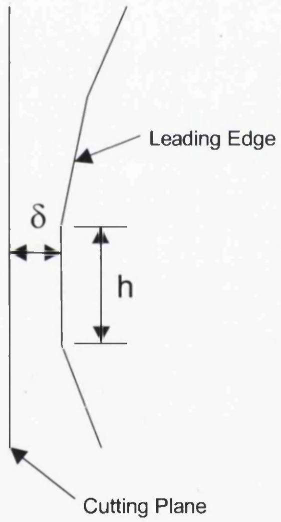


Figure 51 Cutting plane and leading edge mesh problem

In order to cope with this, and to prevent this situation from occurring, a method has been developed that entails inspecting the angle between the normal for the cutting plane and those of the surface boundary triangles, where the angle between these two vectors is less than a pre-defined tolerance then the face is marked. From this information, it is then a simple task to build the regions of the boundary mesh where a cutting plane should not be placed. Inspecting whether the current cutting plane position lies within one of these invalid regions is trivial, and the cutting plane moved into a valid region if required. The algorithm used to determine the invalid, and thus valid, regions is shown below:

```
Procedure findvalidregions
For each boundary triangle:
    Find normal for the current face
    Determine angle, $\theta$, between normal for current face and cutting plane
    If $\theta <$ tolerance:
        Extend or create invalid zone to include minimum and maximum of current boundary face
    Endif
Endfor
For all invalid zones, check that cutting plane does not lie within any zone
If cutting plane lies within an invalid zone, move to closest valid zone, with distance at least "minimum point density" away from closest invalid zone
```

The invalid regions for the Thrust supersonic car are shown shaded in Figure 52.

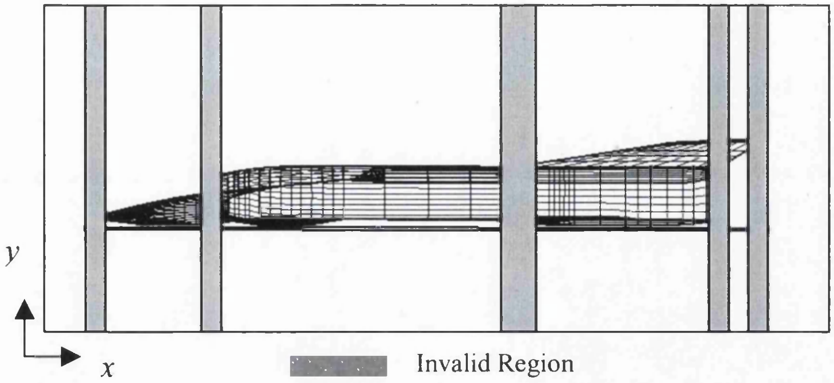


Figure 52 Invalid regions for Thrust Supersonic car testcase

4.3.1.5 Edge Derivation

Once a valid area for the cutting plane has been found, the edges that form the boundary for inter-domain two-dimensional mesh generation, must be extracted from the surface triangulation. The edge data structure is extracted from the surface triangulation and from this dataset a loop of order *n*, where *n* is number of surface triangles in the current domain, is processed and all edges of the triangulation that cross the cutting plane position are extracted. These edges for the EADS Gulfstream, for a cutting plane intersecting the wing, fuselage and engine nacelle are shown in Figure 53.

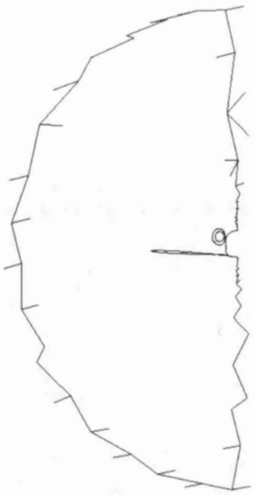


Figure 53 Raw edge extraction for EADS Gulfstream

A number of pre-processing steps are taken before these edges are suitable for two-dimensional mesh generation; the edges must be formed into singly connected loop(s), the path of the boundary edges smoothed in three dimensions and the edges mapped into two dimensions.

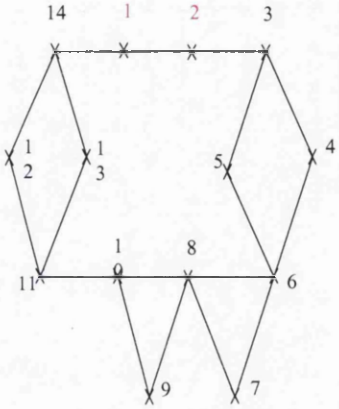
4.3.1.6 Singly Connected Loop Derivation

The edges extracted from the surface triangulation, as shown in Figure 53, are formed into closed correctly orientated loops in order for the inter-domain mesh generation stage to be possible. All nodes within the edge loops can be connected to two edges only. It is clear in the figure that the edges extracted do not fulfil this criterion, and thus pre mesh generation processing must take place.

A loop across the edge connectivity to count the number of nodes connected to each edge can remove the edges that are formed from nodes that are singly connected. Once this step has taken place, the existence of points within the loop that have more than two edges meeting at them requires consideration.

A simple form of weighted graph analysis can quickly and efficiently deal with these edges, and form a closed rim of edges. A start and end point is chosen from the list of edges on each separate loop of edges. This point is selected as a node that is present in only two edges, and thus would not be removed by the weighted graph analysis. Each node is then given a weight by advancing through the loop from start to end. To form the closed loop, it is then necessary to work backwards through the edges from the end point, choosing the lowest weighted node at every point. Marking the edges as the process progresses through the edge loop enables the removal of non-marked edges.

This process is more readily described with a simple edge loop, shown in Figure 54. The start and end nodes are chosen as nodes 1 and 2 (marked in red). A weight is applied to each node, by initialising a counter and incrementing this for every node connected to the current node. The weights calculated for this graph are shown in the table of *ip*(node number) and *iwt*(weight assigned to each node). The table on the right of the figure shows the final path around the edge loop, obtained by moving backwards through the mesh, moving to the node with the lowest weight at every point where a choice is available. The red numbers represent the nodes that have been removed from the edge loop to make the loop singly connected.



IP	IWT	Loop Order
1	-10	1
2	0	2
3	1	3
4	2	4
5	3	6
6	4	8
7	5	10
8	6	11
9	7	12
10	8	14
11	9	7
12	10	13
13	11	9
14	12	5

Figure 54 Simple system of edges to show weighting path

The process applied to the edge derivation shown in Figure 53 is shown in Figure 55.

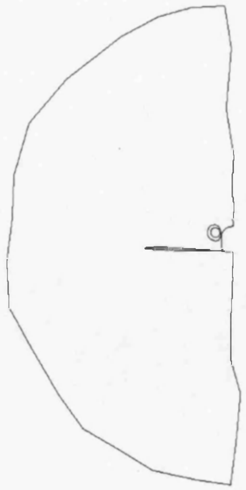


Figure 55 Edges smoothed for Gulfstream

4.3.1.7 Edge Orientation

The edges now form closed loops and must be orientated correctly to ensure that mesh generation stage forms the mesh of triangular elements in the correct regions. Typically, a mesh will be generated on the left hand side of an edge, defined by the edge's direction. The direction of an edge is determined by the connectivity of nodes that forms the edge, thus it is necessary to orientate edge loops so that the triangles are generated in the regions where tetrahedra will exist in the final volume mesh. Figure 56 shows a simplified example to explain this. The problem consists of two hollow cylinders, one with a smaller radius passing through the larger radius. For clarity, the box shaped outer boundary is not shown. On the lower right hand side of the figure, is the regions shown in green where the final tetrahedral mesh would exist. The regions where edges would be extracted for two-dimensional mesh generation are shown in black. For the both cylinders, two loops would be extracted, that represent the inner and outer surfaces. In the region between the outer boundary and the outer surface, the region between the inner surface of the larger cylinder and the outer surface of the smaller cylinder, and the region inside the smaller cylinder, tetrahedral are required.

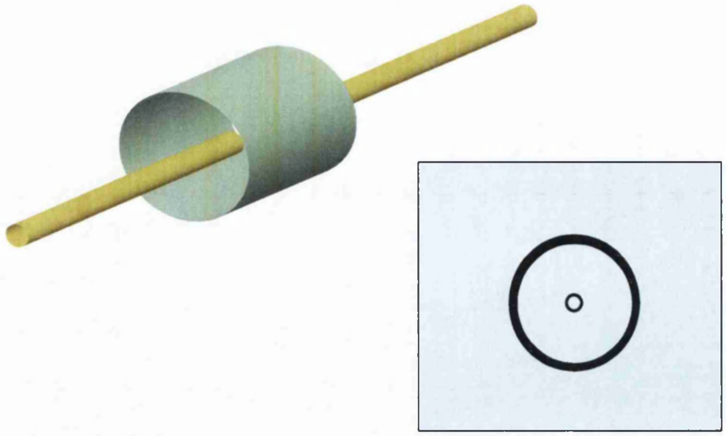


Figure 56 Regions for meshing after edge extraction

With aerospace geometries, cutting through engine nacelles can give edge loops inside other edge loops, and with multi-domain problems the orientation must be able to cope with separate domains. The steps taken to perform the orientation are shown below, with corresponding algorithms.

The edge loop orientation concentrates on identifying which loops exist inside other loops. Two methods exist in determining this; ray-line intersection, shown in Figure 57 where a point from the edge loop under consideration is chosen, and a line drawn to anywhere outside of the bounding box of all of the edge loops. By determining the number of times that this ray intersects the edges in all the loops, the current loop can be identified. In Figure 57 a loop of singly connected edges is shown, with two rays starting from arbitrary points within space. The top ray is outside of the edge loop, and intersects with edges from the loop four times. The ray below this one originates from a point inside of the edge loop, and intersects with the edge of the loop three times. This occurs for any ray that does not pass along an edge, and hence is a valuable tool in determining whether a point lies inside or outside of an edge loop.

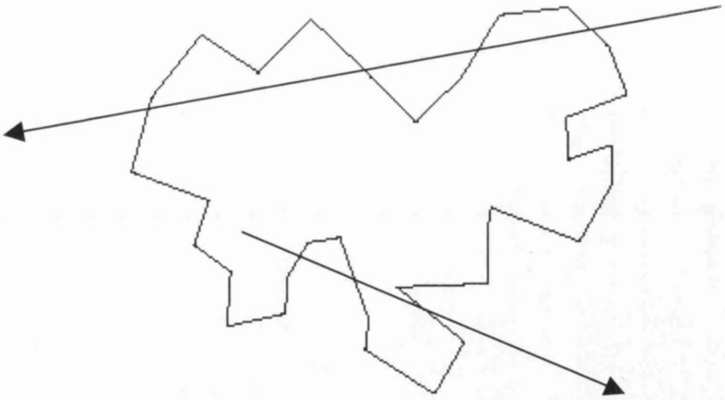


Figure 57 Ray-Line intersection

The second method involves determining the angle between a single node on the current edge loop and the edges of all other edge loops in the domain, Figure 58.

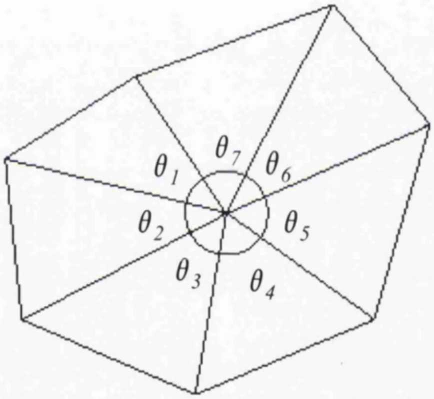
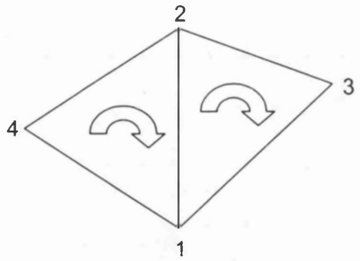


Figure 58 Sum angles method

If $\sum\theta = 2\pi$ then the edge loop under consideration is inside of the other edge loop, if $\sum\theta \neq 2\pi$ then the edge loop is outside of the other loop. The direction of an edge loop can also be determined in this manner, by locating a temporary node at the nodal average position of the edge loops, and determining the total angle as before. If $\sum\theta = 2\pi$ then the edge loop is orientated clockwise, if $\sum\theta = -2\pi$ then the edge loop is orientated anticlockwise.

4.3.1.8 Boundary Face Colouring

The term colouring in reference to mesh generation is used to describe a particular state of the element being marked. Here, colouring is used to identify the partition to which a boundary face belongs. The process splits a domain described by triangular faces into two separate domains, thus the colouring list for the boundary faces will be filled with integers one or two. The surface mesh decomposition process relies on colouring to identify which partition the boundary faces have been sorted into. Since all procedures up to this point have operated on the edges of the boundary mesh, the information from this has to be transferred to the faces. A closed rim of edges forms the boundary for two-dimensional mesh generation, and hence where the inter-domain mesh will be connected. A flood fill procedure has been employed to march out from either side of the edges, marking the colour of each face as the algorithm progresses. The scheme is node based, and the boundary face colour is derived from the nodal colour. Initially, the nodes forming the boundary for inter domain mesh generation are marked, and then the faces that form each edge are marked. This method uses the orientation of the boundary faces to determine which side of the cutting plane a face lies. This method ensures that the faces are marked contiguously, and in order *n*, where *n* is the number of nodes that form the boundary where the inter-domain mesh will be generated. The method is shown more clearly in Figure 59. The edge under consideration has nodes 1 and 2. Element I contains this edge in the correct order as shown by the connectivity table, and so from knowledge of the orientation scheme it is possible to decide which side of the cutting plane this element lies. Element II however, also contains edge 1-2, but in this case in the reverse, hence the element lies on the opposite side to element I.



Element No	Element Connectivity		
I	1	2	3
II	1	4	2

Figure 59 Marking of faces by orientation

Once the faces that will form the inter domain mesh boundary have been marked, the flood fill algorithm can begin. A loop over number of points is used here, to reduce the calculations required. To permit this, a data structure containing the elements connected to nodes is built first, before the flood fill procedure can begin. The flood fill algorithm is shown in Figure 60.

```
Procedure floodfill
        For each node in the mesh:
          If node has been marked:
            For each face connected to current node:
                Mark faces with current node's colour
            Endfor
          Endif
        Endfor
        If no faces have been marked in the pass:
          For each node in the mesh:
            If node has not been marked:
              Determine which side of the cutting plane it lies
              Mark node accordingly
              Restart algorithm at top
            Endif
          Endfor
        Endif
```

Figure 60 Psuedo code for floodfill procedure

Before the colouring of the boundary faces can be accepted, and the two dimensional mesh generation process begins, the edges must be smoothed in three dimensions and checked to ensure that the mapping of the edges into two dimensions does not cause the edges to intersect, and to reduce the chance of intersection between the inter-domain and the original surface mesh. A number of smoothing methods were tested during the development, which included inspecting the angles between given edge paths. The most successful of these methods is described here.

The smoothing of the boundary edges is not a classical smoothing, such as a Laplacian type smoother that moves the nodes to the centre of the ball of nodes, but the path that the inter domain boundary will take is smoothed. The nodal positions in $\Re^3$ are not changed, simply the edges that form the inter domain mesh generation boundaries are changed. This type of smoothing is required since it reduces the possibility that the mesh generated on the inter domain plane when mapped back into three dimensions will not intersect the original domain boundary. By inspecting the edges, and the faces that form the edges, and re-routing the edge path, such that it does not follow around two edges of a given triangle, a smoother path can be created, Figure 61.

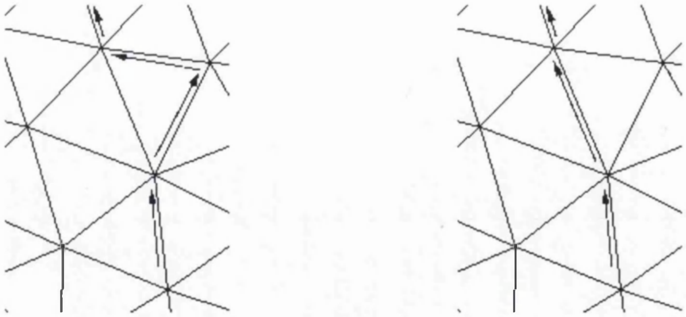


Figure 61 Smoothing of edge paths

Once a set of edges that is admissible for two-dimensional mesh generation has been found, the coordinates of the vertices that form the boundary must be mapped onto the two-dimensional cutting plane. The occurrence of nodes tangling as they are mapped into two dimensions is dealt with by referring to the orientation data, and once the nodes are placed in the correct position, the procedure of two-dimensional mesh generation can begin. Clearly, not only is it possible for edges to become tangled, but also the edges when mapped into two dimensions can intersect with each other. Detecting this is trivial, and moving the position of the plane and beginning the edge extraction loop once more overcomes this problem.

4.3.1.9 Two-Dimensional Inter Domain Mesh Generation

The slave operating on the three-dimensional boundary mapped into two dimensions performs the generation of the inter-domain mesh. Initially this process was performed by the Delaunay mesh generator as incorporated into the three dimensional stage, operating in two dimensions (i.e. z coordinate = 0). However, it was found whilst this was suitable for small test cases, where the inter-domain mesh had a small number of elements, once larger test cases were tried the boundary recovery process in three dimensions began to fail. This was caused by the reduced quality of the inter-domain mesh, which did not exactly meet the specifications as defined by the mesh control entities (sources), due to interpolation performed by the volume mesh generator, when generating the sequential mesh. To alleviate this, an advancing front procedure was used to generate the mesh in the parametric space, using a background mesh defined by the sources.

The background mesh was found to be a way in which it was possible to accurately and automatically determine the point density at any given position within the domain. Before any parallel mesh generation can begin, the pre-processor must be used to generate the background mesh, which consists of connecting the sources to create tetrahedra. The background mesh is generated using a coarse triangulation of the box that encloses the domain, and inserting points into this mesh at the source positions. The mesh is generated using a Bowyer-Watson procedure. An example of the background mesh is shown in Figure 62. A Cutting plane is used through the mesh to show where the tetrahedral are created. The inner points in the mesh represent the centre of the sources used to control the mesh point spacing.

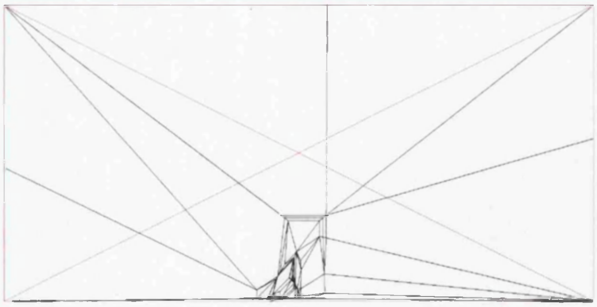


Figure 62 Background mesh for Dassault Falcon geometry

4.3.1.10 Mesh Validity

The inter-domain two-dimensional mesh is checked for validity in the original sub-domain before acceptance. In the development stage of the parallel mesh generator, the inter-domain mesh when inserted into the sub-domain could intersect with the original boundary. Initially this occurred due to the inaccuracies in determining the correct point spacing near to the boundary edges. Using the background mesh method to determine the mesh point spacing at any point as described in section 4.3.1.9 this problem was alleviated. However, it is still possible for the two surface meshes to intersect, in particular where an edge has been removed from the boundary of the inter-domain mesh boundary in order to produce closed loops.

If the inter-domain mesh intersects with the sub-domain mesh a number of options exist. Point insertion is possible, in an attempt to 'lift' the inter-domain mesh away from the original surface.

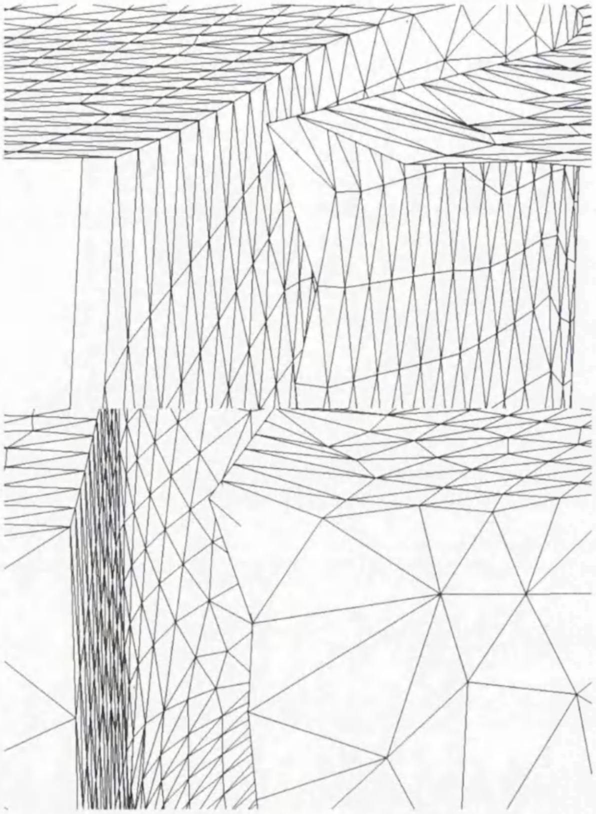


Figure 63 Intersecting inter-domain and original mesh with point insertion

Figure 63 shows an example of an inter-domain mesh intersecting with the original boundary. At the top of the figure, the boundary for two-dimensional mesh generation can be seen, where one side of the mesh has been removed. Generating the inter-domain mesh and placing into the surface mesh, in the lower of the figure, shows the intersection with the original boundary. Iterative point insertion by splitting the edges, and mapping the new node onto the plane, can alleviate this problem.

Figure 64 shows a two-dimensional example, where the points used to define the boundary of the inter-domain mesh are some distance away from the cutting plane position.

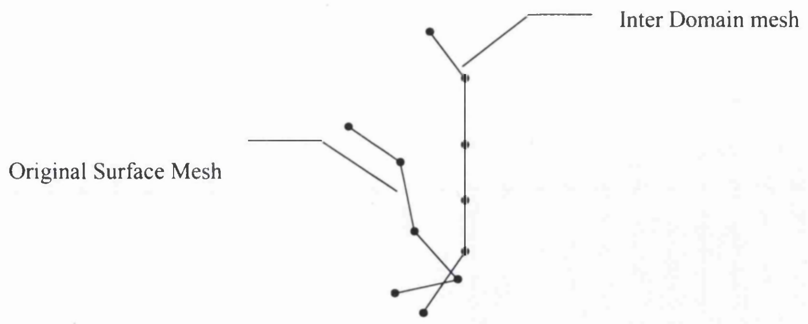


Figure 64 Intersecting inter-domain boundary in two dimensions

This results in the regions connected to the original boundary highlighted in red, where intersection between the inter-domain mesh and the original boundary could occur. Inserting points into the mesh along the edges that are intersecting, it is possible to alter the profile of the inter-domain mesh, such that the mesh is moved away from the original boundary, removing the intersection, Figure 65.

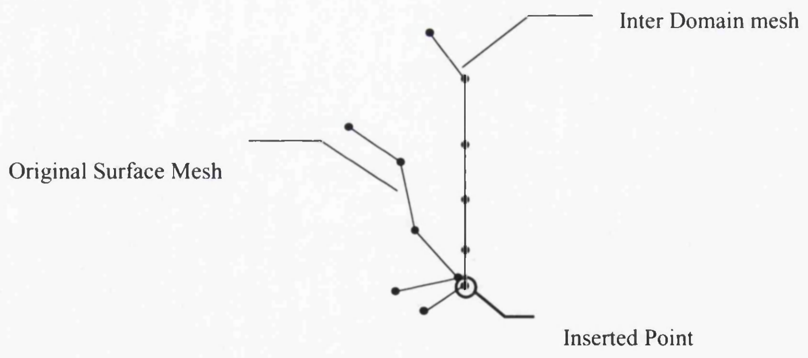


Figure 65 Point insertion to alter the mesh profile

Once point insertion has failed to remove the intersection, the plane can be moved and the inter-domain mesh generation process started once more.

4.3.2 Load Balancing for Volume Mesh Generation

Once the domain decomposition stage is complete, the volume meshes consisting of tetrahedra must be generated. At the parallel mesh generator inception, the sub-domain volume meshes were generated and left as a partitioned mesh, that together with the communication information, form the complete mesh of the domain and are ready for the simulation stage of the computational simulation cycle. Hence, it is necessary to know a-priori to generating the volume mesh how many processors will be available for the solution stage. In this situation, where the mesh will be left partitioned, the partitions produced by the domain decomposition step can produce unbalanced sub-domains.

The use of the partitions derived for mesh generation creates a number of significant computational problems for the simulation stage. The communication size of a mesh, a measure of the number of communication nodes that adjoins the partitioned meshes, can significantly affect the performance of a parallel algorithm, in particular one whose algorithm is a fine-grained algorithm, which typifies fluid dynamics and electromagnetic parallel solvers.

The partitioned meshes from the domain decomposition step can be highly unbalanced, in particular for computational fluid dynamics meshes where the edge length changes through the computational domain. In this type of calculation, the output from the parallel mesh generator in partitioned form is not suitable for computation in terms of load balance and communication. This imbalance in load affects the mesh generators ability to generate meshes on computer platforms that have a low overhead of memory; primarily what the parallel mesh generator was created to do.

In an effort to alleviate this problem of unbalanced partitions of volume meshes, a method of over-decomposition and assimilation is used. By automatically sub-dividing the boundary surface mesh into more partitions than are required, and then assimilating these surfaces meshes into the required number of partitions, where the number of boundary faces per sub-domain has a better balance, the volume mesh stage can produce mesh partitions that exhibit improved load balance

A method has been devised to assimilate the sub-domain surface meshes into fewer partitions with lower variance in boundary triangle count. Initially, the method used was to assimilate the domain by inspecting the neighbours of each sub-domain and evaluating the sum of the boundary triangles of a resulting sub-domain created by the assimilation of the two partitions into one. Two partitions are said to be neighbours if they share common boundary triangles, and the two neighbouring sub-domains can therefore be easily assimilated into a single sub-domain by the removal of these shared triangles. This method was found to be unsuitable as the difference between the required number of sub-domains and the total number of over-decomposed sub-domains grew.

A variation on this method entails building the graph that represents the domain over-decomposition. A node is placed at the centre of each sub-domain, and an edge is formed by connecting two nodes (that represent two sub-domains) where the two sub-domains share common faces. Figure 66 shows a simple two-dimensional domain that has been decomposed using the scheme described previously.

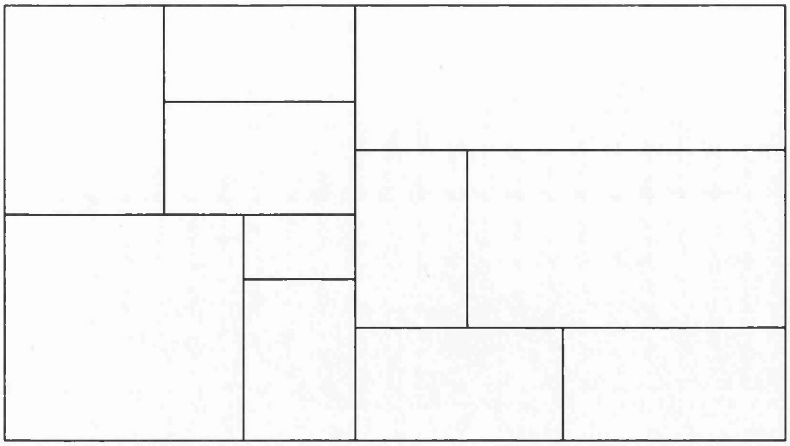


Figure 66 Two dimensional domain decomposition

The resulting graph of this decomposition is shown in Figure 67. By performing a graph analysis on this, using freely available software such as METIS [69], it is possible to obtain the colour of each node in the final decomposition. A typical graph of the decomposition of an F16 geometry and domain is shown in Figure 68.

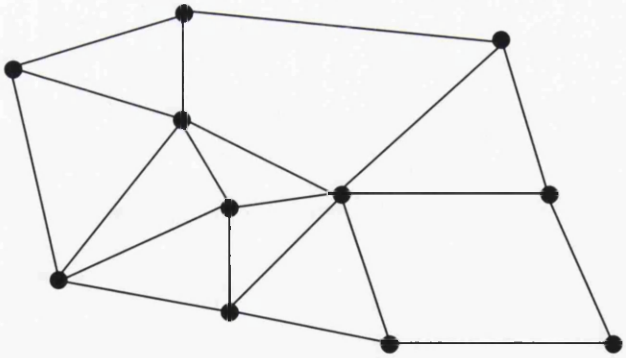


Figure 67 Resulting graph from decomposition

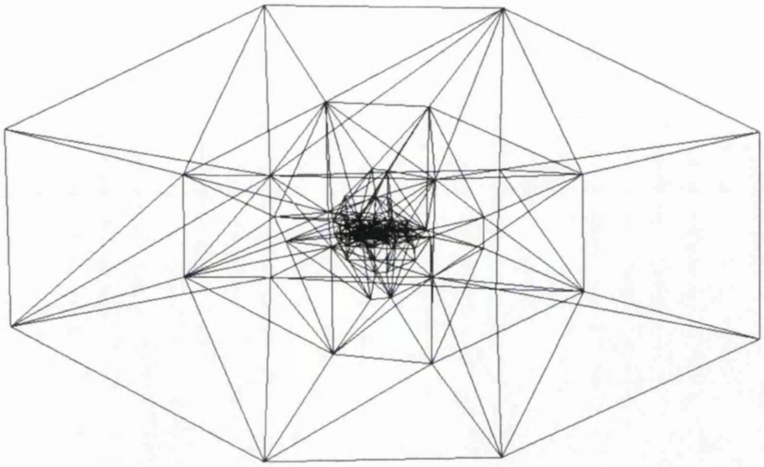


Figure 68 Graph of decomposition of an F16 simulation model

From this information, the sub-domains can be assimilated to fewer partitions with improved load balance in terms of number of boundary faces. Clearly, using this graph without any regard to the quantities that are represented by the edges and nodes will not provide any of the improvements that are possible with graph methods, and so weights are applied to the nodes and edges of the graph. The node weights are set as the number of boundary faces in the sub-domain that the node represents, and the edge weights are defined as the number of inter-domain boundary faces between two neighbouring sub-domains that the edge represents. By using this weighted graph, and graph partitioning software that can partition weighted graphs, it is possible to minimise the edge cut, where all cut edges will represent inter domain boundaries, and thus communication in the final mesh. In this manner, the communication of the final partitioned volume meshes can be reduced, in addition to the better work load balance afforded by applying the boundary face weights. Figure 69 shows the communication costs for the differing graph partitioning methods available through the METIS library. The corresponding load balance in terms of number of boundary faces is shown in Figure 70. This shows that for this testcase, series three, produced using "metispartgraphrecursive" has the lowest communication cost.

Series	Method	Maximum communication nodes
1	Metispartgraphkway	2998
2	Metispartgraphvkway	2566
3	metispartgraphrecursive	2213

Figure 69 Table of Communication costs for different assimilation techniques

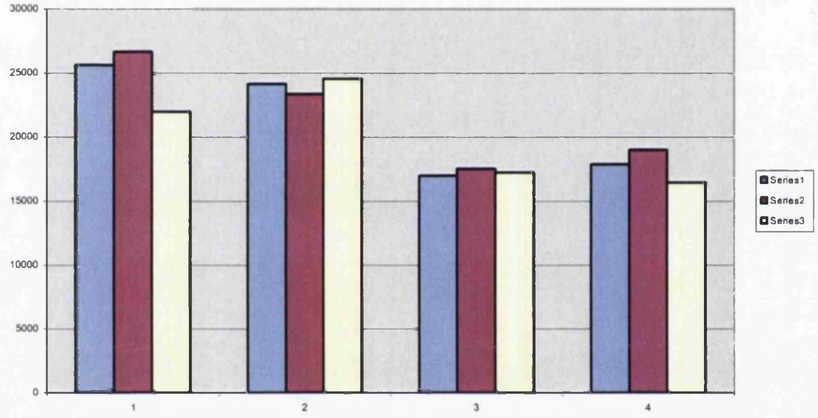


Figure 70 Graph to show number of boundary faces per sub-domain for different assimilation techniques

Figure 71 shows the load balance for a domain decomposition of the EADS Gulfstream into 4 sub-domains. Series two depicts the load balance for an over-decomposition to 64 sub-domains; series three shows the over-decomposition to 128 sub-domains and series four an over-decomposition to 256 sub-domains. The graph shows the improvement in load balance over that provided by a non over-decomposed domain, series one. Although the load balance has been significantly improved by performing this method, the partitioned meshes are not suitable for performing a simulation due to the load imbalance.

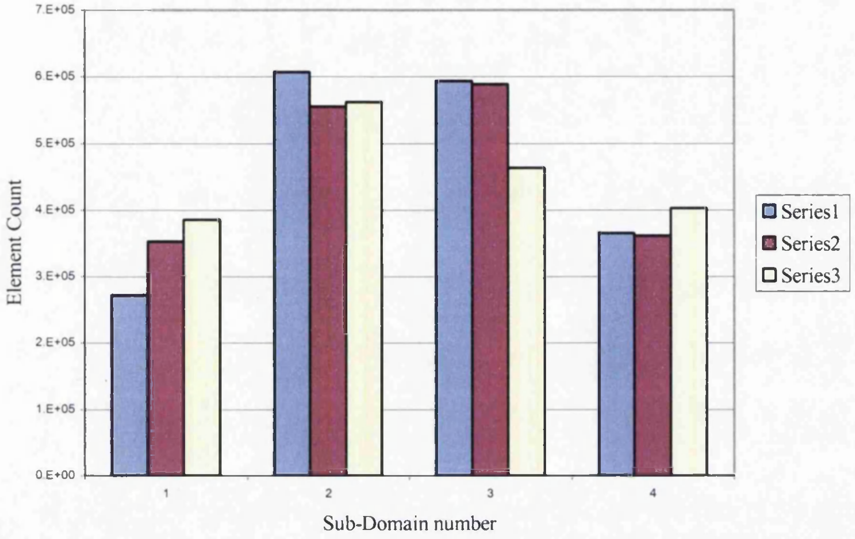


Figure 71 Load Balance for 4 Sub-Domain Gulfstream Decomposition

Figure 72 shows the data logging tool provided by the MPICH message passing library, using the output from the over-decomposition of the Gulfstream mesh to 64 sub-domains and assimilation to 4 sub-domains. The master process is shown in red as process number zero, and all slave processes (in this case eight) are shown in blue. From this output it is evident that the domain decomposition took just under half of the overall wall clock time to decompose the mesh into 64 sub-domains and assimilate into eight. Evident also, is the parallel domain decomposition, as the mesh becomes divided into more sub-domains the load balancing hands out the tasks to the waiting processes.

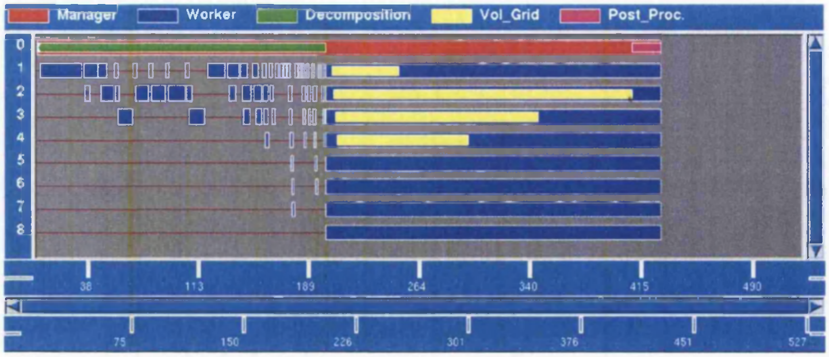


Figure 72 MPE Data logging for 64 domain over-decomposition

Figure 73 shows the output of the MPE logging facility for the Gulfstream over-decomposed to 128 sub-domains and again assimilated to 4 sub-domains. Here the blue blocks represent where the slave process is working on data. The yellow blocks represent the volume grid generation stage. Using eight slave processes and one master process, the domain decomposition uses seven processes to decompose the domain into 128 sub-domains. The gaps that are present during the decomposition phase are due to the method that has been employed with task farming. The tasks are distributed amongst the available processes, and data retrieved at the end. This means that a new loop of decomposition cannot begin until the master process has received all the data. In addition, during the volume grid generation phase, the four processes 5 to 8 are not used since only four volume grid generation tasks exist. Within the domain decomposition step, it is clear that all processes are being used to divide sub-domains, whereas in Figure 72 the last process did not contribute to the domain decomposition step. The volume mesh stage, highlighted by the yellow bars, shows a different pattern to that in Figure 73, due to the assimilation of the sub-domains producing a different result. From the wall clock time scale, it is also evident that the total time for domain decomposition is slightly longer than for the 64 sub-domain case, although the overall wall clock time for the entire meshing process is less than for the 64 sub-domain case.

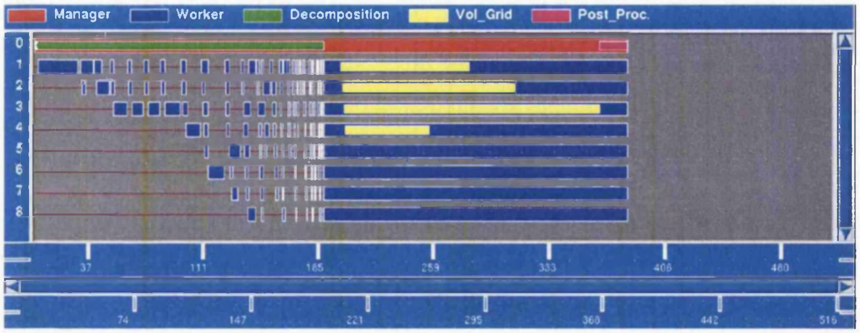


Figure 73 MPE Data logging for 128 domain over-decomposition

The meshes generated in the examples shown in Figure 71 contain a total of 1.8 million elements. These meshes are small enough in size to generate sequentially, and so the parallel mesh generator would not conceivably be used. Figure 74 shows the robust nature of the method, by applying the technique to a complex geometry, the EADS F16 in full store configuration. Here, the domain has been over decomposed to 256 sub-domains, and assimilated to 8 sub-domains for volume meshing. Figure 75 shows the MPE output for the mesh.

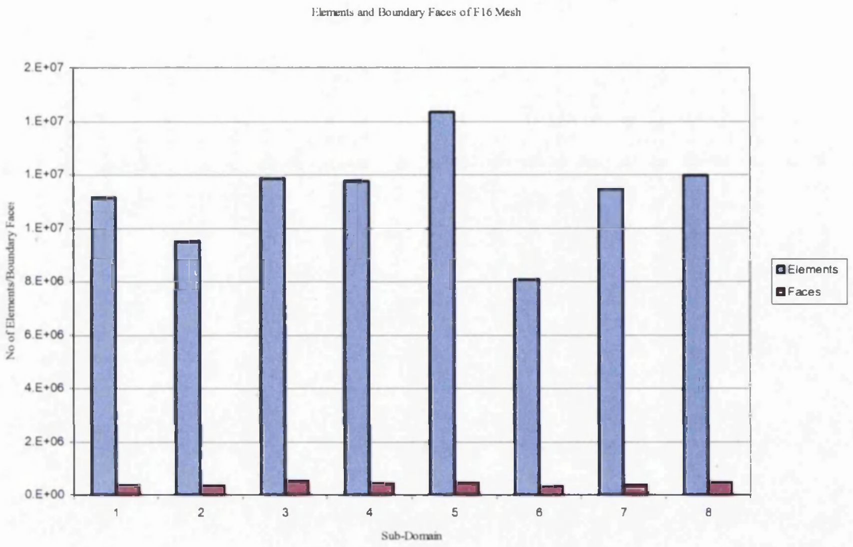


Figure 74 Load Balance for 8 Sub-Domain F16 Decomposition

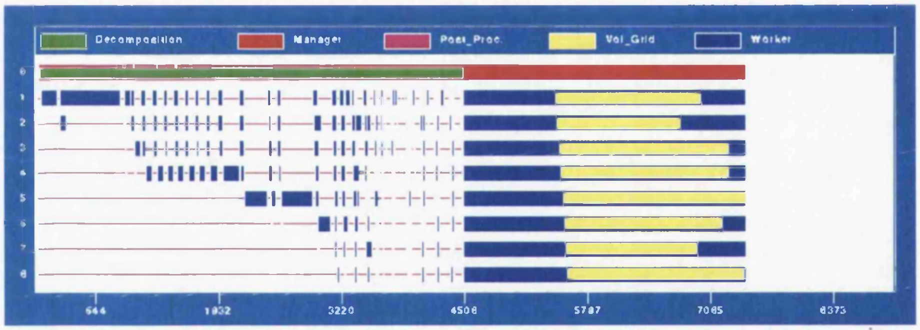


Figure 75 MPE Data logging for 256 over-decomposition of the F16 Geometry

It would be advantageous then to uncouple the dependency between the output of the parallel mesh generator and the input of the solver. Thus a single mesh file is created once all the volume meshes have been completed. The surface mesh assimilation still has a role to play, however, in improving the efficiency of the parallel mesh generator, in as much as reducing the speed paradigm for parallel computations ‘only as fast as the slowest horse'. By balancing the load in this manner, the volume grid generation task is balanced among the processes, such that the time taken is approximately equal for each task. This benefits the memory overhead per process required. In addition to this, the assimilation provides a justifiable step, to ensure that the volume mesh partitions are large enough to reduce the occurrence of bad elements. A surface mesh decomposition that contains many small sub-domains could conceivably reduce the final volume mesh quality

4.3.2.1 Graph Methods for Load Balancing

Mesh partitioning plays an important role in parallel simulations. A poorly partitioned mesh can exhibit unbalanced workload and a large communication cost. Graph methods, that represent the mesh as a graph, either by using the nodes and edges of the mesh as the graph, or using the barycentre centre of every element as a node and the neighbouring element information as the edges, represent a method that can partition meshes into well-balanced partitioned meshes with minimised communication costs. Generally freely available, METIS[69], CHACO[70] and JOSTLE[71] are some of the graph partitioning tools available. A discussion on the merits and improvements to the graph partitioning methodology can be found in Hendrickson et al[72].

4.3.2.2 Surface Mesh Assimilation

To join two sub-domains that share an inter-domain boundary a number of arrays require updating. The master process holds an array of length *TotalNumberOfFaces* and width two. In this is stored the colour of each boundary face, both inter-domain and original. A boundary face is recognised by having a single entry in the first column, the second column containing a zero. An inter-domain boundary face is identified by having two positive entries in the array, each of which signifying the two sub-domains that the face separates.

The output from the graph-repartitioning tool is the colour of each sub-domain for the new smaller number of sub-domains. Isolating the sub-domains into the new groups, and removing the inter-domain boundary faces that join two sub-domains allows the new sub-domains to be created. Finally, once all the inter-domain mesh boundary faces have been numbered accordingly, the boundary faces are renumbered without the faces that have been removed to assimilate the domains.

4.3.3 Volume Mesh Generation

The volume mesh generation stage consists of two steps:

- Orientation of the sub-domain boundary triangles
- Triangulation using the Bowyer-Watson method (including automatic point insertion, boundary recovery and mesh cosmetic routines)

The original boundary surface of the domain will be orientated in the correct direction however; the inter-domain meshes may not be orientated in the correct way. Indeed, as the number of sub-domains increases, it is possible for sub-domains to be constructed from wholly inter-domain meshes.

In order to orientate these sub-domains, a ray-tracing algorithm is used, as described in two dimensions in Section 4.3.1.7. Since the inter-domain meshes are orientated along an axis (and the axis that a face is generated on can be found be inspecting the normal) then the orientation of a face can be found. It would not be efficient, however, to perform this ray-tracing algorithm for each boundary face within the sub-domain. Instead, a method has been employed that orientates the boundary faces by using a single orientated boundary face. The orientation of a domain from a single boundary face consists of determining the direction of each edge in relation to that of the orientated face. Careful programming of this allows the sub-domain to be orientated in the correct direction, even when a sub-domain contains the connection of more than two faces at an edge.

4.3.4 Post Volume Mesh Generation Processes

4.3.4.1 Parallel Mesh Cosmetics

Early on in the history of computational simulation, merely obtaining a mesh that accurately represented the problem proved a distinct challenge and many methods were devised to discretise the computational domain, some of which are described in Section 2. As the solution techniques become more complex, the discretisation of the domain becomes crucial in finding a solution that minimises any errors introduced due to the mesh – in as much as the mesh is a compromise between computing power and resources versus error. A Delaunay triangulation method as described in Section 2.2, with iterative automatic point insertion, requires a post-processing step of the mesh to improve the quality of the mesh. This is required because it is possible for a traditional

Delaunay method to admit 'slivers' and other poorly formed tetrahedral elements into the mesh, as shown in Figure 76 (comes from [7]). The figure shows two boundary triangles {1,2,4} and {2,3,4}, which due to numerical round off, have been connected to form a tetrahedron {1,2,3,4}. In two-dimensions, a set of three almost co-linear points could conceivably form a flat triangle. However, the circumscribed circle would more than likely include another element that will then allow the removal of such flat triangles. In three-dimensions, four co-planar points could, due to numerical round off, form a flat tetrahedron. Here, the position of a nearby point could be outside of the circumsphere described by the flat element, and thus the tetrahedron (known as a sliver) will be left within the mesh.

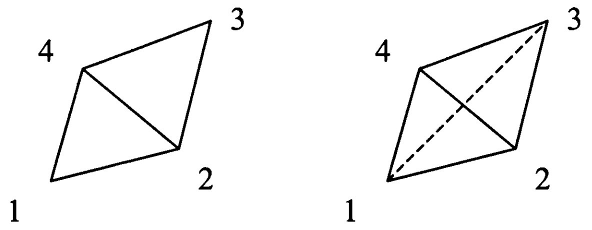


Figure 76 Slivers generated by Delaunay mesh generation

In order to improve the mesh quality of a sequentially generated mesh, three methods are used; edge swapping, element collapsing and mesh smoothing (Laplacian).

Edge swapping entails swapping shared edges between two neighbouring elements in the case of two dimensional mesh generation, and in three dimensions swapping shared faces. Figure 77 shows the two-dimensional case of swapping an internal edge between two neighbouring triangles. Clearly, this has removed the badly formed elements that were admitted by the Delaunay criterion and maximised the minimum internal angle in the two elements.

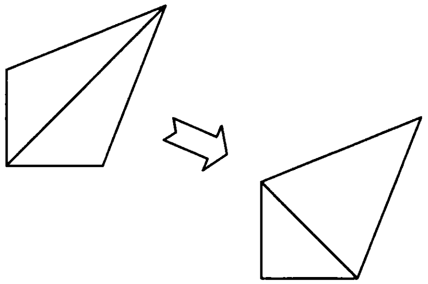


Figure 77 Edge swapping in two-dimensions

When an element has been generated that has a small internal angle, and cannot be repaired by swapping edges or faces, then that element should be collapsed, and the resulting hole filled with a local re-triangulation. Figure 78 shows that the ill-formed element that, for the purposes of the example, cannot be repaired by swapping has been removed from the triangulation by moving one of the nodes that forms the flat element. Essentially the node in the centre of the diagram has been moved to lie on the lower edge.

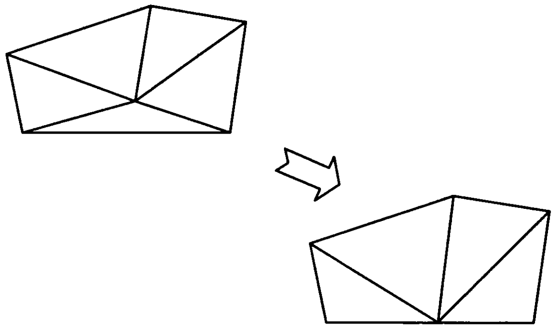


Figure 78 Element collapsing and local retriangulation

Once the edge/face swapping and element collapsing routines have completed, the mesh should also be smoothed to ensure that the point density, and thus edge length, varies smoothly through the computational domain. Laplacian smoothing provides a means by which the mesh can be smoothed efficiently, by placing the nodes of a mesh at the centre of the ball formed by the elements to which the node is connected. Assuming the edges of the mesh are represented by springs, minimising the strain within the springs iteratively, with the force exerted by a spring proportional to its length and along its direction [50]. Iteratively moving points until the points are moved less than a pre-defined tolerance produces a mesh that is smooth.

The mesh cosmetics schemes described previously are applicable to a sequentially generated mesh, and can then also be used during the parallel mesh generator, in the aim to provide a good quality mesh. A mesh generated by the parallel mesh generator should demonstrate close to the same quality indicators when compared to an equivalent sequential mesh.

The parallel mesh will, however, have planes within the final mesh that are clearly identifiable as points that were generated during the decomposition step. These planes within the mesh cause problems for the cosmetics routines, due to the methods which are used to improve the quality of the mesh; in particular when an element is wholly formed of surface nodes, the quality operations of smoothing, element collapsing and edge swapping cannot improve the element quality. Whilst the occurrence in a sequential mesh of this is rare, the decomposition stage and the method of planar cuts increases this occurrence dramatically. This is more readily described in two dimensions, Figure 79, where the nodes generated on the plane are marked in red. The mesh has been mapped back into two dimensions (for the two-dimensional case shown here), with newly generated nodes placed on the plane of the cut. The rim that formed the boundary for the inter-domain mesh generation step is not on the plane, and thus the Delaunay criterion has allowed for a triangle to be created that is formed by the three boundary points. Any swapping of the shared edge produces two elements of decreased element quality and so the element is passed over without alteration.

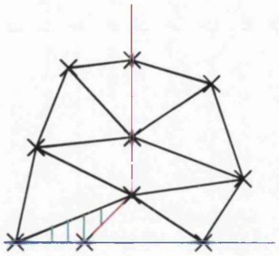


Figure 79 Inter domain element quality problem

Clearly, it is possible to improve the quality of this element, because in the neighbouring partition the inter-domain node also exists. Thus, the inter domain node can be moved and the elements surrounding this region can be operated on by the quality enhancement procedures.

The simplest way to perform cosmetics on the completed parallel mesh would be to perform the cosmetics sequentially. However, this could require a substantial amount of memory to be available to a single processor, which violates a specification of the parallel mesh generator, to reduce memory requirements. To overcome this, and to have only a single mesh loaded per process, a parallel scheme has been developed to perform the cosmetic routines in parallel on the partitioned meshes. By passing elements across inter-domain boundaries, a given sub-domain can be enriched with elements from it's neighbouring partitions, which then allows the cosmetic routines to remove the badly formed elements which arise from the inter-domain boundary. Before any elements can be passed across and shuffled between neighbouring sub-domains, it is necessary to perform a global numbering stage. The stage is described in detail in Section 4.3.4.1.1.

The neighbouring sub-domain data is found from the surface number of boundary triangles, which is set as a negative number of the opposing sub-domain when the sub-domains were built. Initially, a pseudo-parallel scheme was devised, that utilised two slave processes of the parallel mesh generator. One slave is known as an extract slave, which purely extracts elements from a required sub-domain, and sends these elements to the other slave, the cosmetic slave. The master, who hands out the tasks sequentially to the cosmetic slave, controls the overall process. The structure is shown in Figure 80.

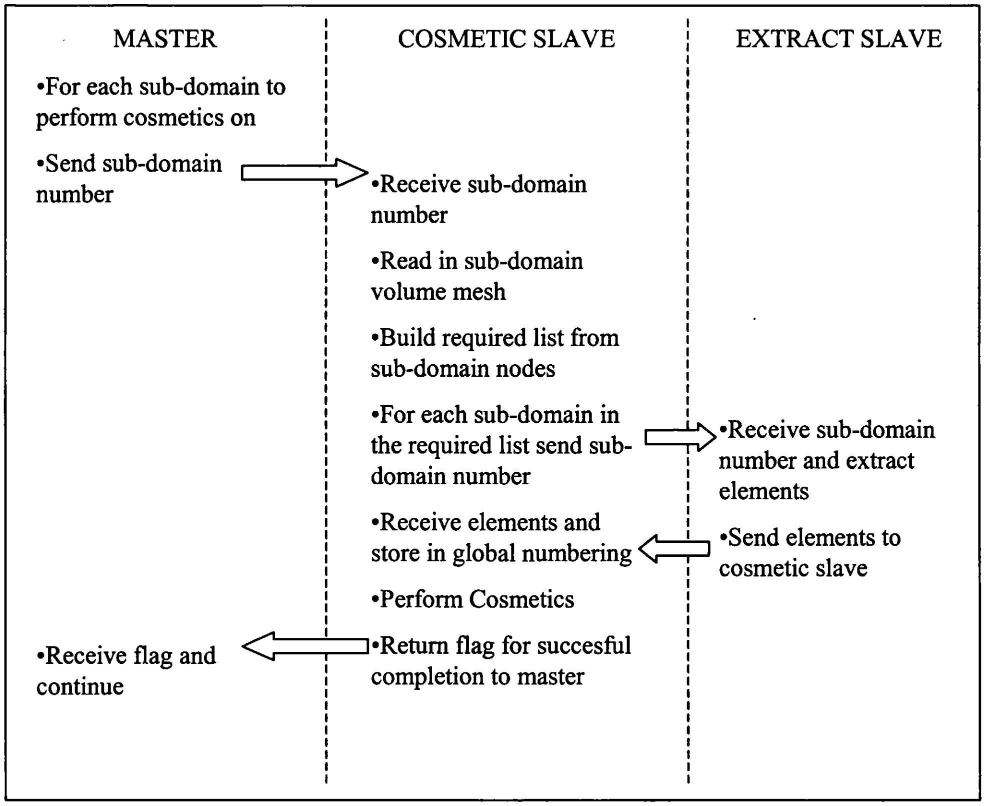

MASTER
COSMETIC SLAVE
EXTRACT SLAVE
•For each sub-domain to perform cosmetics on
•Send sub-domain number
•Receive sub-domain number
•Read in sub-domain volume mesh
•Build required list from sub-domain nodes
•For each sub-domain in the required list send sub-domain number
•Receive sub-domain number and extract elements
•Receive elements and store in global numbering
•Send elements to cosmetic slave
•Perform Cosmetics
•Receive flag and continue
•Return flag for succesful completion to master

Figure 80 Psuedo-sequential cosmetics structure

The master process sends a message to the cosmetic slave containing the number of the sub-domain to perform the cosmetics procedures on. Upon receiving this information, the cosmetics slave opens the globally numbered mesh file, and reads in the sub-domain mesh. By looping over the boundary face surface numbers, it is possible to build up the list of sub-domains that neighbour this sub-domain through faces of tetrahedra. Looping over this list and sending the neighbouring domain number to the extract slave, allows the extract slave to retrieve from the neighbouring sub-domain all the elements that are connected to the sub-domain that is under consideration for cosmetics. To ensure that all elements are passed to the sub-domain, the global node numbers are used and the extract slave is permitted to enrich the list held by the cosmetic slave so that sub-domains that are connected by edges and points are also included in the element extraction process. Figure 81 shows the case of using boundary points for receiving elements, if boundary edges (faces in three-dimensions) have been used the elements connected by a single node in sub-domain 4 would have been ignored. Clearly, this element is required to accurately smooth the elements in the region where the four partitions meet, and so enriching the list and using global node numbering allows for the element to be included in the cosmetic routines.

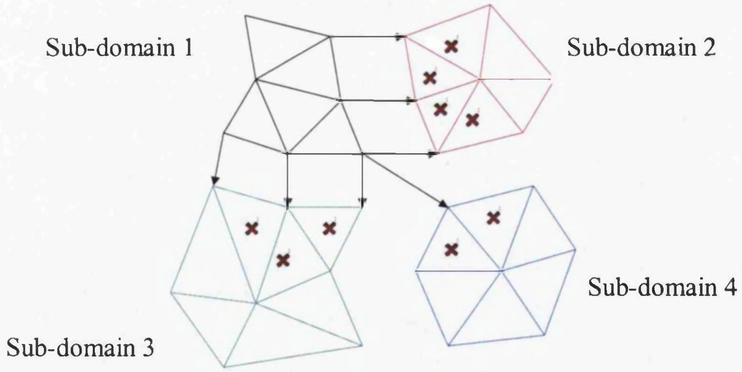


Figure 81 Two dimensional element extraction example

The extract slave must also create a valid sub-domain from the remaining elements, in terms of boundary face data, since this domain will also have the cosmetic procedures applied to it. Once the cosmetic slave has received all the elements from connected partitions, the mesh is updated with the new elements and boundary face data. Once this has been completed, the cosmetic routines described previously can be applied and the resulting mesh written to file.

A method that uses all the available processes has also been developed. The structure of this is shown in Figure 82. Here, the problem is set-up by identifying the independent tasks that exist within the domain decomposition. These independent tasks exist because it is not permitted to perform cosmetics on the same node/element concurrently. Hence, two sub-domains that require elements from sub-domains that are not common can perform the cosmetics procedures.

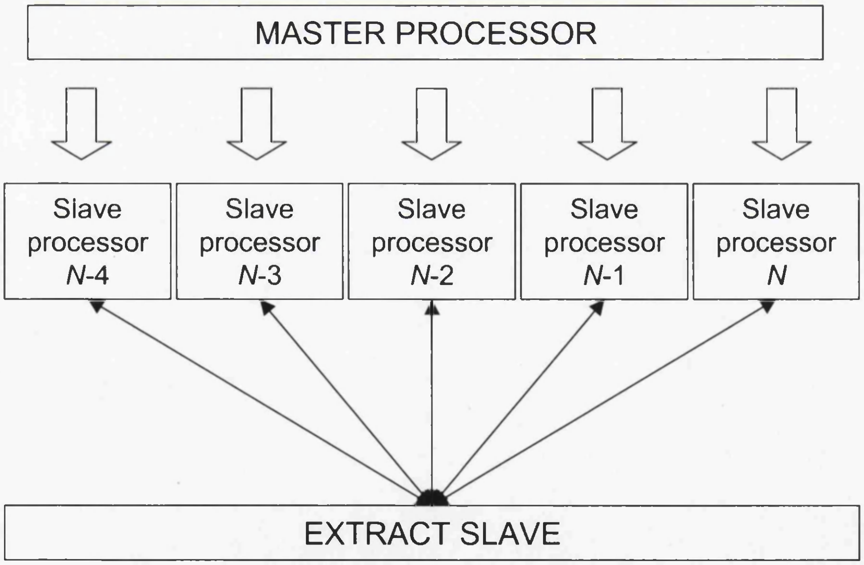


Figure 82 Parallel cosmetics process structure

Figure 83 shows simple two-dimensional domain decomposition; the sub-domains highlighted in yellow are those that can be operated on without interference. Hence once a single loop has been processed, only two sub-domains will have been altered by the mesh cosmetics procedures. Hence, in order to process all the sub-domains, the loop must continue to identify opportunities for parallelism until all the sub-domains have been processed. Dynamic load balancing is required to deal with an imbalance of tasks and processes.

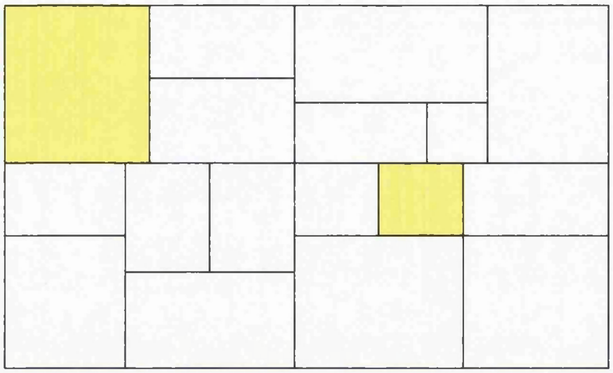


Figure 83 Two dimensional domain decomposition showing independent tasks

4.3.4.1.1 Global Numbering Scheme

The global number of each node within the sub-domain mesh files must be determined before the cosmetics routines and element passing routines can begin. The global numbering will start with the original surface mesh numbering, and all extra nodes added as they are generated. Hence, after the surface nodes, the inter-domain nodes will be added. The master process builds the sub-domain boundaries in the local numbering system for each sub-domain, and maintains a record in a single dimension array of the global numbers of these nodes. It is simple and efficient then to number the volume nodes globally, starting with the first sub-domain and then proceeding sequentially in the order of sub-domain numbering.

The result of this type of numbering scheme is that areas of the mesh are numbered contiguously. This differs to a sequentially generated Delaunay mesh, where new nodes are inserted into elements with automatic point insertion so two new points could be at different ends of the mesh. For large-scale computations, the mesh is typically too large to read into a single processor. Before the simulation can begin, a mesh partitioning scheme is required to balance the load and minimise communication across the available processors. In order to read a mesh onto a distributed computer that exceeds the memory available to a single processor, a scheme has been developed [73] that allows a number of processors to read separate sections of mesh.

Here, for an *N*-processor case containing *M* elements and *P* points computation, each processor reads in *N/M* elements and *N/P* points and then communicates the nodal coordinates to the respective processor. Since the mesh is numbered contiguously, reading in a mesh generated by the parallel mesh generator is considerably quicker than that generated by a sequential nature due to the 'pockets' of contiguously numbered elements throughout the domain.

4.3.4.1.2 Element Extraction Criteria

It is important to maintain valid partitioned meshes at all times, to enable the restarting of the mesh generator. Once a sub-domain has had elements removed or added by the corresponding slave the boundary faces, the faces of elements that form the boundary, must be extracted along with surface number information. In order to determine the elements that form the boundary, it is necessary to build the neighbouring elements information.

The outcome of this routine is an array of length *NumberOfElements* and width four. In each of the four columns is stored the neighbouring element, where each column represents the face number as defined by the tetrahedron local numbering scheme. A loop of order *NumberOfElements* can then extract the boundary face data wherever a zero in a column occurs.

The extraction of elements is based on a node marked scheme. Initially, the nodes that are required are marked, and the node to element connectivity array built for the mesh. By looping over the nodes required, all elements that are connected can be marked for removal. In this manner, the required node list can be updated interactively through the loop, which allows for a smooth layer of elements to be extracted and multiple layers. Anomalies within the boundary definition of an object to be meshed are propagated through the mesh, in both unstructured and structured meshing. Figure 84 shows regions of poor quality elements caused by poor boundary definition and surface definition. These regions, circled, cause poorly formed tetrahedra to be formed during the generation process. These elements can cause problems for the Delaunay process as the further points are introduced into the mesh, causing the error that originated at the geometry definition stage to propagate through the surface mesh to the volume mesh. For this reason, it may not be possible for the cosmetics procedures to remove badly formed elements simply by receiving a sigle layer of elements surrounding the element under consideration.

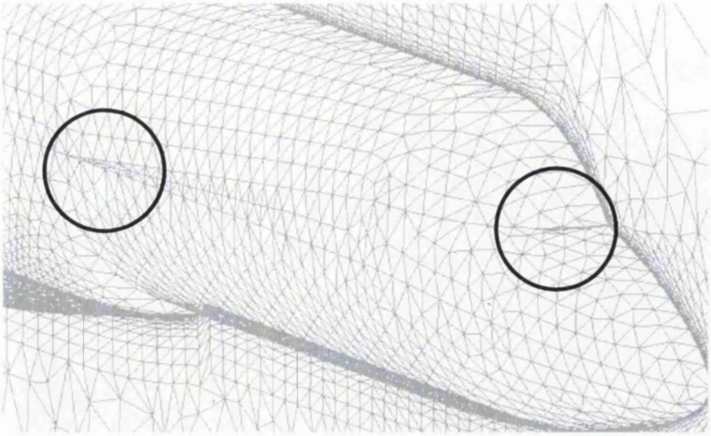


Figure 84 Error propagation through the mesh in unstructured meshing

To overcome this, it has been found that differing number of elements should be passed between neighbouring sub-domains. Consider a two-dimensional mesh generated in two separate sub-domains, Figure 85, the inter-domain shared edges are highlighted in red. Initially, sub-domain one requires elements to be received from sub-domain two. Since sub-domain two has not been processed by the cosmetics routines with extra elements, two layers of elements are requested and sent to sub-domain one. Once sub-domain one has been processed, sub-domain two must have the same procedure applied to it. However, since two layers of elements have already been sent across, it is necessary to request four layers of elements from sub-domain one.

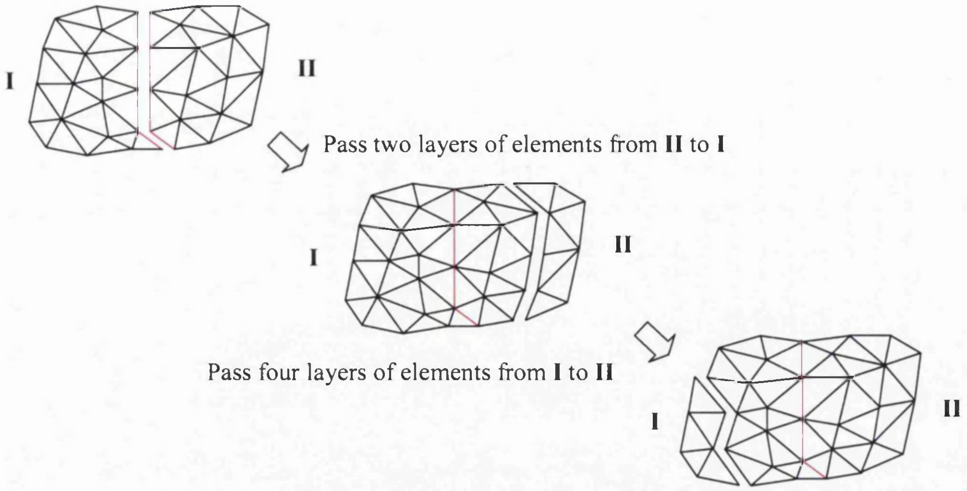

I
II
Pass two layers of elements from II to I
I
II
Pass four layers of elements from I to II
I
II

Figure 85 Two sub-domain element passing example

In this manner, all the elements in the region of the cutting plane are smoothed, and any errors that may have propagated are removed

4.3.4.1.3 Mesh Comparisons

The results for the parallel cosmetics procedure are shown in Figure 86 and Figure 87 for computational electromagnetic simulation geometry of a perfectly electrical conductor (PEC) sphere. The quality of the meshes generated by the parallel mesh generator are obtained from the final completed mesh, and compared to an equivalent sequentially generated mesh, using the same volume mesh generator in the sequential and parallel models. Figure 86 shows the comparison of dihedral angle between a sequential mesh, a parallel mesh with two sub-domains without parallel cosmetics and a parallel mesh with two sub-domains with parallel cosmetics. The same cosmetics parameters were used for all three meshes, and it is evident that the mesh generated without cosmetics is different from that with the cosmetics. A similar test for mesh quality with a larger number of sub-domains, Figure 87, shows a similar situation. The performance of an electromagnetic solver is largely based on the time step calculated from the smallest element height in the mesh. Table 1 shows the results of running the solver on these meshes, and demonstrates the height of the smallest element and the resulting time step.

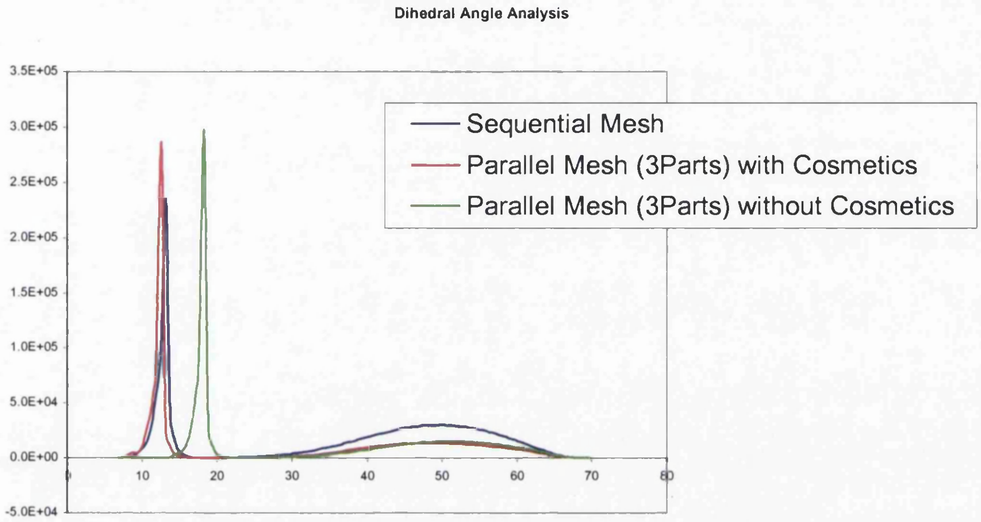


Figure 86 Results of parallel cosmetics for 2 partition domain decomposition

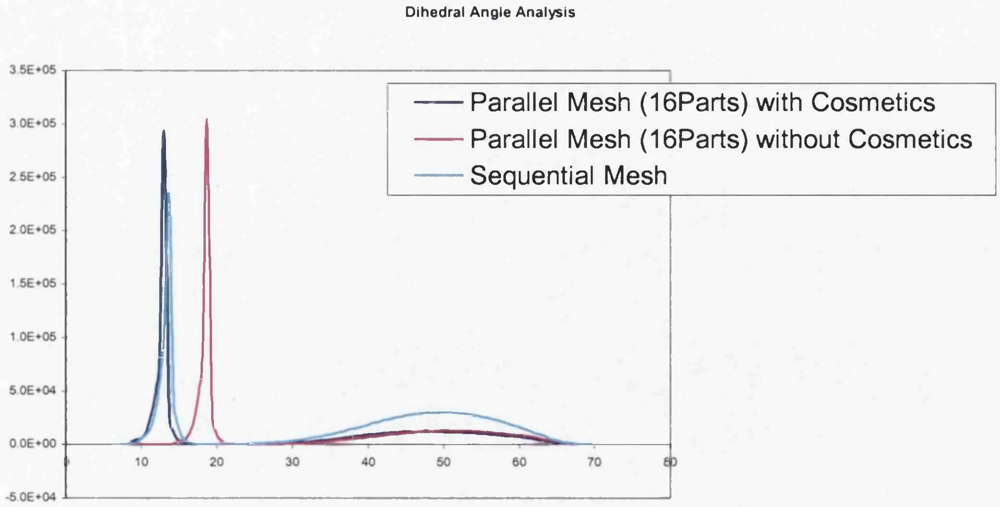


Figure 87 Results of parallel cosmetics for 16 partition domain decomposition

Mesh	No of Time Steps	Minimum Element Height
Sequential Mesh	660	0.76169E-02
3 sub-domain mesh with cosmetics	660	0.759-7E-02
3 sub-domain mesh without cosmetics	34750	0.13352E-02
16 sub-domain mesh with cosmetics	1125	0.43085E-02
16 sub-domain mesh without cosmetics	34750	0.13352E-02

Table 1 Element Height and Resulting time step for PEC sphere

4.3.4.2 Final Mesh Options

In order to perform a numerical simulation on the mesh generated in parallel, the final meshes must be output in such a form that is compatible with the input of the particular solver in question. If the solver incorporates a mesh-partitioning tool, the desired output would possibly be a global mesh of the domain; however, if a mesh-partitioning tool is not incorporated, then the partitions generated by the parallel mesh generator could be used. In the case of partitioned meshes being used, a communication table is required, from which the solver can extract the list of common nodes between partitions.

4.3.4.2.1 Communication Data Retrieval

Once the volume mesh generation stage of the parallel mesh generator has completed, and any cosmetics having been performed on the resulting sub-domains, the communication table containing the common nodes must be extracted. For each sub-domain volume mesh file, a communication table file is generated that contains the local node number for each node on the surface of the sub-domain, and the corresponding local node number in all domains that share this node. This is shown clearly with a two-dimensional example of a partitioned mesh, Figure 88. Partition I (left) has five communication nodes, {1,9,10,12,11}, that are common with nodes in partition II (right), {6,10,9,8,7}.

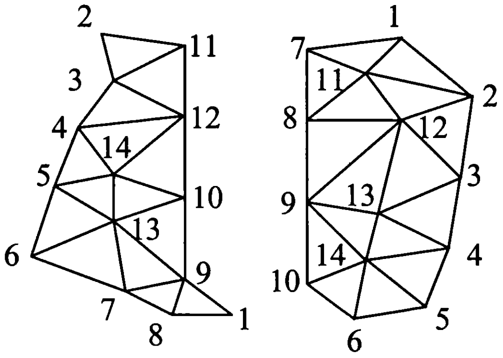


Figure 88 Two-dimensional communication example

4.3.4.2.2 Global Mesh Completion

As an alternative to using the domain decomposition used for volume mesh generation, the partitioned mesh can be brought together to form a single global mesh file. In order to keep within the specifications of the mesh generator, not more than one sub-domain mesh file can be present on a processor at an instant in time. To achieve this, a loop of order *NumberOfSubDomains* is processed. At each step in the loop, the corresponding sub-domain mesh is opened and loaded into memory. If the mesh has had parallel cosmetics performed on it, the element and boundary face connectivity exists in the global numbering, so the data need simply be written to the file, and any new points added correspondingly. If the mesh is not in global numbering, the sub-domain is numbered correctly and output to the global mesh file.

4.4 Multi Physics led Extensions to the Parallel Mesh Generator

The parallel mesh generator has thus far been limited to the generation of isotropic tetrahedral elements. Unstructured inviscid flow simulations require isotropic tetrahedral elements throughout the domain to resolve the flow solution accurately. A typical inviscid flow simulation over an aircraft geometry would require of the order of one million nodes, or 5 million tetrahedra. As described in Chapter 3, the use of anisotropic elements to reduce the number of elements for a given simulation whilst retaining the solution accuracy is desired. The anisotropic Delaunay method discussed earlier presents significant problems for optimisation in terms of parallelisation and indeed extension to three dimensions. The advancing layer method of anisotropic mesh generation has been used to provide the meshing for boundary later simulations in a number of mesh generators developed previously [74][75]. The optimisation of this procedure is required before it is suitable for inclusion into the parallel mesh generator. Herein, various optimisation methods are discussed. The following is the description of the addition to the parallel mesh generator to generate anisotropic hybrid elements for high Reynolds number flow simulations, and semi-structured tetrahedral elements to truncate the infinite domain of electromagnetic scattering simulations.

4.4.1 Semi-Structured Layer Generation

4.4.1.1 Sequential Methods

Section 3 described the use of anisotropic elements for complex flow field simulations. The methods are described in terms of solution resolution for sequential generation schemes. The extension of the parallel grid generator to include an option to include a layer of anisotropic elements within the mesh increases the flexibility of the program, such that it can be used to discretise geometries for complex flow simulations.

4.4.1.2 Advancing Layer Scheme

The advancing layer technique to generate anisotropic elements was first published by [76]. The method is based on the advancing front technique but a number of subtle changes are used to ensure the layers of anisotropic elements are formed. In the advancing front method, the front grows from a boundary from the shortest edge into the unmeshed domain. Conversely, in the advancing layer method, a complete layer of element is grown from the boundary at a time. Therefore, a second layer of elements will only be generated once the first layer has been completed, subject to geometry constraints. The point placement scheme in the advancing front scheme typically places points where the new edge lengths generated will closely match the lengths of the existing edges of the mesh, Figure 89.

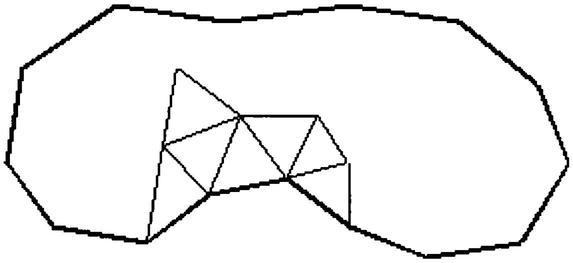


Figure 89 Point placement scheme for Advancing Front Method

Point placement in the advancing layer scheme is controlled by user definition of the spacing of each layer of the mesh. The nodes are generated along extraction lines, which are defined at normals from the mesh surface. These normals are determined for each node in the surface mesh above which a boundary layer will be generated, by averaging the normals for each surface elements that surround the node, Figure 90.

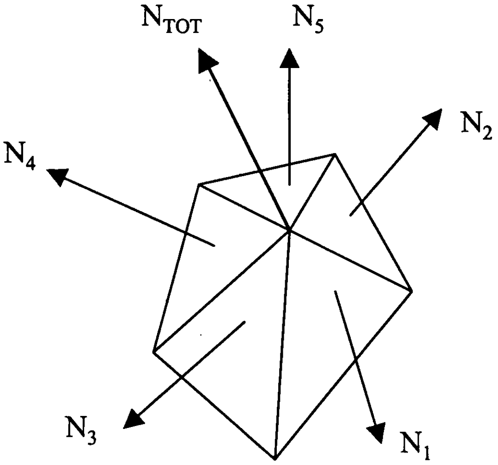


Figure 90 Normal averaging for nodal values

The advancing layer method for a simple two-dimensional geometry is shown in Figure 91. Starting at a node on the surface, a new node is generated along the extraction line created from the node. The distance along the extraction line is controlled by user input, and defines the height of each layer in the boundary layer mesh. The elements are created in the same manner as the advancing front method, but the advancing layer method ensures that the a new layer of elements is not started until all the nodes have been tested for new element creation.

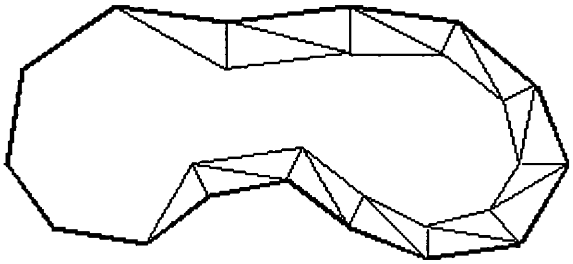
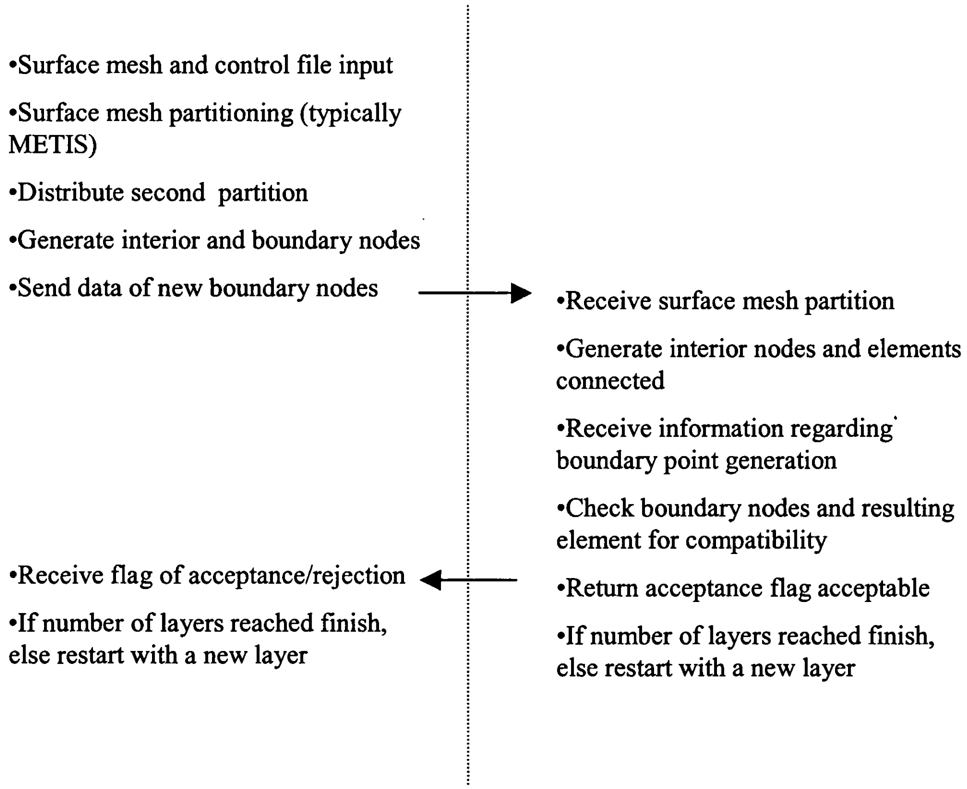


Figure 91 Point placement scheme for Advancing Layer Method

4.4.1.3 Optimisation

The parallel mesh generator described previously can be thought of as a parallel harness, into which any sequential unstructured mesh generator can be incorporated to generate large unstructured meshes on parallel platforms. The sequential Delaunay mesh generator developed previously [18] has been used for the work shown here. This sequential mesh generator has the option to generate anisotropic elements for viscous flow simulation using the advancing layer procedure. In a typical mesh of an aerospace geometry suitable for high Reynolds number flow simulations, the anisotropic elements within the viscous layer can account for over 50% of the entire mesh elements. To date, meshes for high Reynolds numbers flow simulations containing up to 100 million tetrahedral elements have been generated, placing large demands on memory from the computing power available. It would be advantageous then to generate this boundary layer region in parallel, in order to reduce the memory overhead requirement to acceptable levels, and to speedup the generation process.

Initially, a method to generate the anisotropic elements in parallel was devised. As with most unstructured mesh generation methods, the advancing layer technique does not easily lend itself to parallelisation. Any method would require a fine-grained parallel algorithm, and thus be heavily dependant on communication, requiring a computer with low latency inter-process communication. The program structure for such a parallel advancing layer structure, for a two-process computer, is shown in Figure 92.



- •Surface mesh and control file input
- •Surface mesh partitioning (typically METIS)
- •Distribute second partition
- •Generate interior and boundary nodes
- •Send data of new boundary nodes
- •Receive flag of acceptance/rejection
- •If number of layers reached finish, else restart with a new layer

- •Receive surface mesh partition
- •Generate interior nodes and elements connected
- •Receive information regarding boundary point generation
- •Check boundary nodes and resulting element for compatibility
- •Return acceptance flag acceptable
- •If number of layers reached finish, else restart with a new layer

Figure 92 Program structure for parallel advancing layer method

Once the master process has read the surface mesh of the computational domain, a decomposition of the mesh is required to partition the domain into *NumProcs-1* sub-domains, using METIS. This surface mesh is only the surfaces upon which the advancing layers will be grown from, an example of which is shown in Figure 93, for a surface mesh of the EADS Gulfstream geometry. The different colours for the edges represent the different sub-domains.

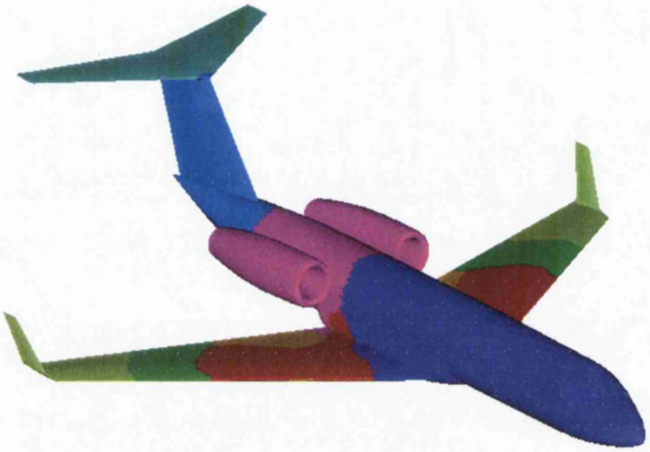


Figure 93 Partitioned surface mesh for viscous parallel generation

Distributing these sub-domains to the respective processes, allows for the advancing layer process to begin. In order to avoid the case where one partition will have generated a new node, and not been generated within the second partition, a hierarchy is placed on the processes. The hierarchy ensures that a node is not considered for generation by more than a single process. Distributing the partitioned surface mesh, the boundary nodes of each sub-domain are recorded and owned by a single process. A boundary node is considered by the process with the lowest number connected to the node, and the result communicated to the remaining domains that share this node. The lowest process number in this case will generate the node and connect to other nodes within the sub-domain to generate the tetrahedral elements. The partitioned surface meshes will contain all triangles belonging to the sub-domain, and all elements connected to the boundary nodes, as shown in Figure 94 for a two sub-domain decomposition case. Thus, a layer of elements will be connected to the sub-domain that are there to obtain the normals for the nodes, entailing duplication of nodes and elements, known as ghost elements.

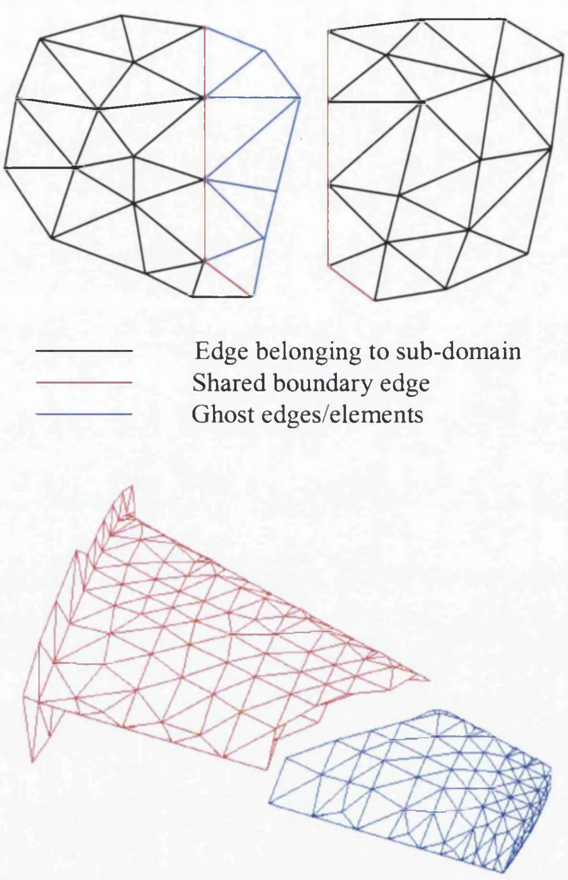


Figure 94 Two sub-domain showing ghost cells for compatibility

Upon receiving the information regarding generation of a boundary node, a domain will form the elements surrounding the node, and check for any intersections with the current front within the domain. Once a point has been accepted, a message of acceptance is sent to the process that generated the node. This procedure is repeatedly until either the number of layers required has been met, or no further elements can be generated due to intersections with the original domain boundary or the newly generated viscous mesh.

This type of scheme restricts the parallel mesh generator to machines with low latency and high bandwidth in order to maintain program efficiency. This method was developed further, into a coarse grained approach more in keeping with the scheme of the parallel isotropic mesh generator.

Generating the inter-domain partitions prior to generating the volume allows the isotropic parallel mesh generator to be coarse grained, and therefore able to operate on computer platforms with low bandwidth, such as networks of workstations. A similar approach was devised for the advancing layer approach. The inter-domain meshes in this case consist of three-dimensional triangles originating from the surface mesh attached at the join between two sub-domains. Figure 95 shows the procedure applied to the EADS Gulfstream; Figure 96 the procedure applied to the M6 Wing, where the anisotropic triangles separating the domains are shown, known as viscous fences.

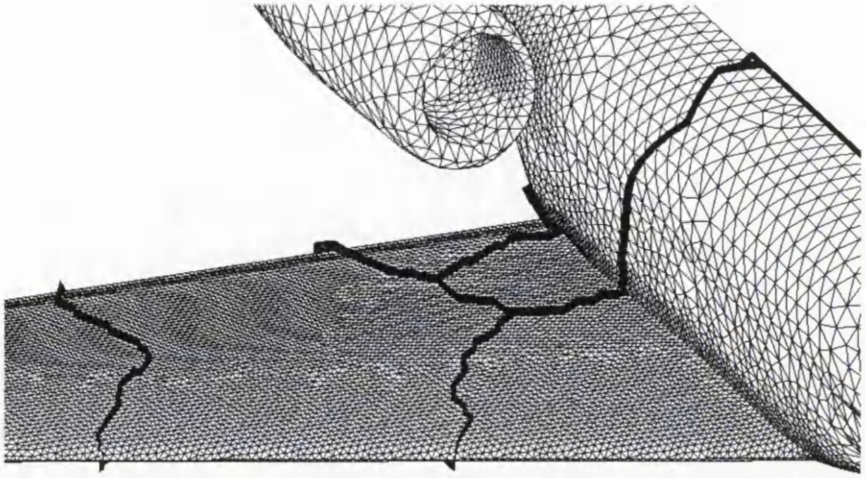


Figure 95 Viscous Fences for the EADS Gulfstream

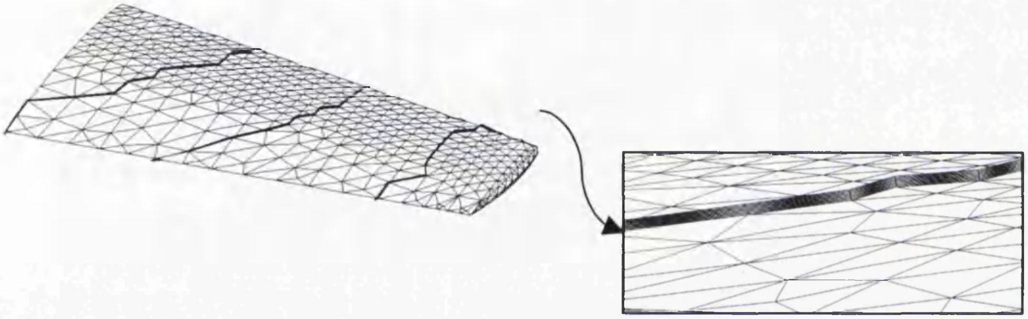


Figure 96 Viscous Fences for M6 Wing

The master using the same procedures as the tetrahedral element procedures generates the viscous fences. The only change is made to the connectivity in order to generate the triangles. Once this has been performed, the partition meshes along with the corresponding fences can be sent to the processes for advancing layer tetrahedral element generation. This reduces the communication to zero between the slave processes, and thus the advancing layer generation method and the isotropic parallel generation method have the same process structure.

However, this method works for simple geometries such as the M6 wing, where all surfaces are regular and do not fold back, so any intersection between viscous layers cannot occur. For more complex geometries, it is possible for viscous layers to intersect, and thus after each layer of elements has been generated a communication step to broadcast the top surface of all individual viscous meshes is required to check for intersection. Whilst this scheme reduces the memory required to generate the viscous mesh, the time penalty was too large a price to pay.

A different method was sought that allows the memory overhead to be reduced and the time penalties to be kept to a minimum. It was decided that the most suitable method to generate the viscous layers in a manner that would reduce the memory overhead was to optimise the advancing layer method itself. Since each layer of elements is based upon the preceeding layer's top face, then outputting the previous layer and freeing the memory could significantly reduce the total amount of memory required. Hence, the advancing layer method was incorporated into the parallel mesh generator sequentially, with each layer of elements written out to disk, and corresponding memory freed. The time increase for the I/O operation whilst still a factor, is smaller than that of the increases caused by the previous two parallel advancing later methods described previously. This optimisation allows that only a sinlge layer of boundary layer elements are held in memory at any given time.

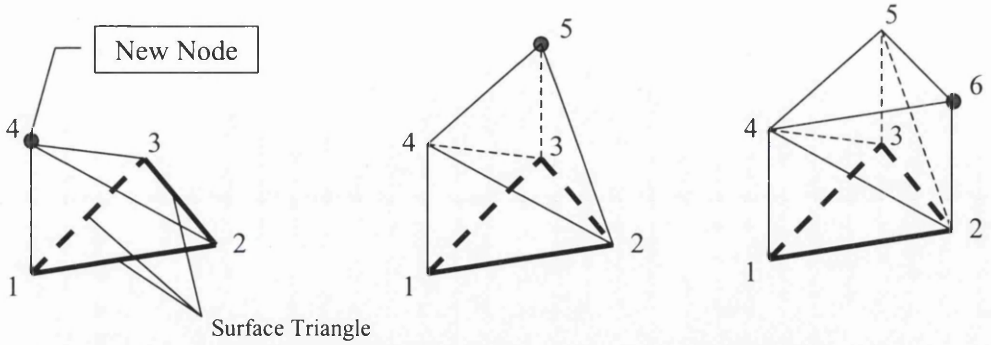
4.4.2 Computational Fluid Dynamics Application

4.4.2.1 Boundary Layer Region Mesh Provision

The advancing layer technique is more comprehensively described in [21]. A brief overview is given here, highlighting the optimisation of the procedure and the subsequent extension for hybrid element generation.

Figure 97 shows the process for generating tetrahedral elements once a point has been successfully placed along a surface normal originating at a point on the surface mesh. An array of length *NumberOfSurfaceNodes* is maintained throughout the generation process that contains the last node to be generated on each normal. Once a new node has been generated, the tetrahedra containing this point must be generated. This is performed by creating for each surface triangle that contains the normal upon which the node was generated a tetrahedron, by connecting the top node of the three surface points and the new point.

During the optimisation of this procedure, the nodes for the new layer are generated and checked for intersection with the current front (the top faces of the element layers). The new faces of the tetrahedron are inserted into a tree data structure [8], the array of nodes above surface nodes updated and the tetrahedron connectivity deleted. The elements are created 'on-the-fly' when writing out each layer, and so it is possible to generate prism, pyramid and tetrahedral elements when writing to the file. A prism element is created wherever all three nodes of a surface triangle have a node generated above them (element on the right of Figure 97), a pyramid element where only two of the three nodes have a node above them (element in the middle of Figure 97) and a tetrahedron whenever only on of the three nodes has been generated (element on the left of Figure 97).



Element	Local Connectivity			
	1	2	3	4
I	1	2	3	4
II	2	3	4	5
III	2	5	4	6

Figure 97 Element creation for advancing layer method

Once the boundary layer region has been successfully meshed using this approach, the isotropic region must be discretised with tetrahedra. By extracting the top surface of the viscous layer, and merging with the outer boundary and any symmetry planes that exist, the boundary surface mesh of the isotropic mesh generator is created. This surface mesh can then be handed to the parallel mesh generator for partitioning and meshing as described previously. Once the isotropic region has been meshed, the volume mesh is completed and placed into a single mesh file, ready for partitioning and solution.

The results and comparisons between the parallel and the sequential viscous mesh generators are shown in Section 5.1.

4.4.3 Computational Electromagnetics Application

4.4.3.1 Perfectly Matched Layer Region Provision

Standard low order finite element methods for solving the linear Maxwell equations for electromagnetic simulations place large demands on the mesh generation process. The frequency of the electromagnetic wave and the size of the object determine the point density and hence the size of mesh required. Figure 98 shows the mesh requirements for the simulation of the scattering of an electromagnetic wave across an aircraft of length 20 metres. Radar frequency simulations of interest to the aerospace industry start at around 8 GHz, and hence the mesh required to perform such a simulation would be in the region of 100 million elements.

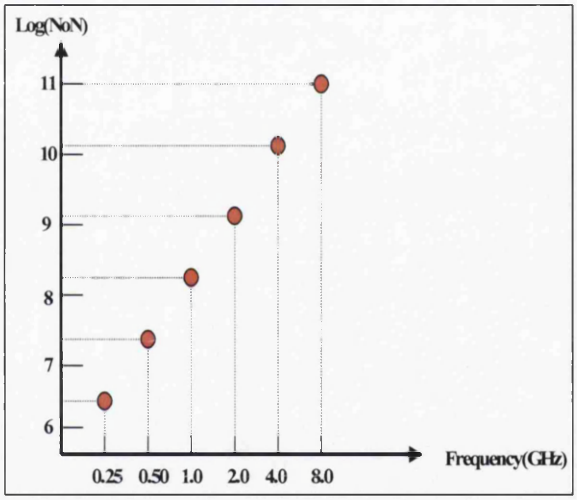


Figure 98 Graph of mesh requirements for scattering of an electromagnetic wave across 20 m aircraft

The generation of such large meshes is not possible using standard sequential methods, and hence a parallel scheme is employed. The infinite domain can also be truncated by the use of a semi-structured region of elements, known as the Perfectly Matched Layer (PML). In order to retain the memory savings provided by the parallel mesh generator, the PML region is required to be generated in such a way that these savings are maintained. Using the advancing layer method, this time operating on the outer boundary surface, with constant layer height defined as the mesh spacing on the boundary it is possible to discretise the region with isotropic semi-structured layer. The formulation for a PML was devised by [77] [78] and prevents the reflection of the wave back into the computational domain. The optimised advancing front scheme is employed in a similar manner to the viscous generator, although here the regeneration of the isotropic region boundaries is not required.

An example of the method, applied to the classical electromagnetic scattering testcase of a PEC sphere is shown in Figure 99. The slice through the mesh clearly shows the PML region of semi-structured mesh expanding from the isotropic region, ten layers thick. Also evident is the proximity of the outer boundary to the object, afforded by the PML region.

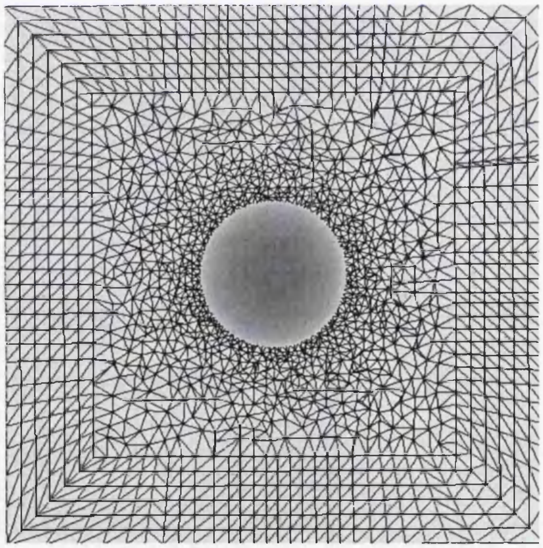


Figure 99 PEC Sphere slice through mesh

5 Parallel Mesh Generation Results

5.1 CFD Results

5.1.1 M6 Wing Geometry

A single ONERA M6 wing is a geometry used regularly in aerospace to compare results of CFD modelling and wind tunnel tests. The geometry was used in the development of the parallel mesh generator, as a simple testcase. As stated previously, the primary aim of the parallelisation of the mesh generator is to increase the size of problem possible, and hence it is necessart to compare the memory usage of the parallel mesh generator, and that of the sequential mesh generator used with the harness. Previously, the boundary layer meshing algorithm of advancing layers was described, with the enhancement to reduce memory requirements. Generating ten layers of anisotropic elements on the M6 wing geometry, shown in , the optimised advancing layers method used a total of 14MB of memory, compared to a non-optimised memory requirement of 17MB. In total, the memory usage for generating the complete mesh (isotropic and anisotropic parts) for the parallel grid generator is 108MB compared to 201MB for the sequential method.

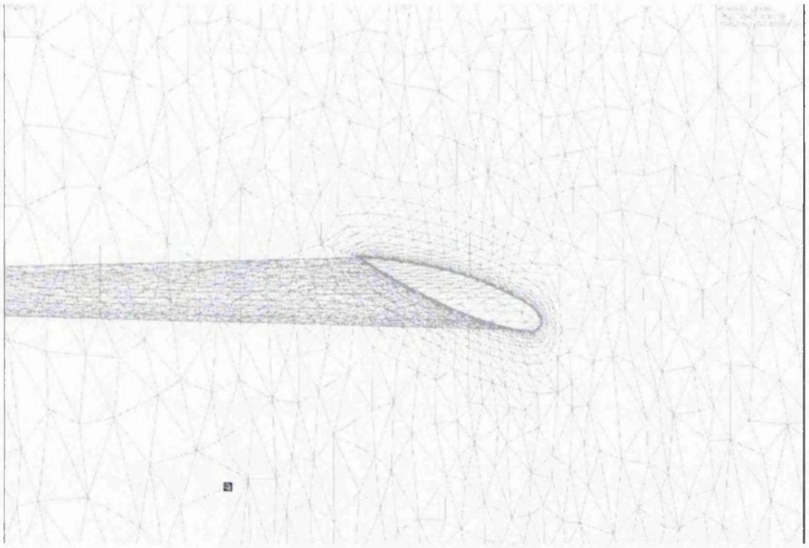


Figure 100 ONERA M6 wing anisotropic mesh

5.1.2 F15 Military Aircraft Geometry

Two simulations were performed with inviscid airflow at a speed of Mach 0.85 for the F16 military aircraft geometry. The “finger of four” formation is routinely used as a typical flight pattern, and so is a legitimate simulation to perform. The computational domain was set with the four aircraft in a cylindrical domain. In order to gain optimal computational efficiency from both the parallel mesh generator and the fluid dynamics solver, the over-decomposition method was used. Initially the computational domain was decomposed into 512 sub-domains, and then assimilated to 8 sub-domains for volume meshing. The workload for each sub-domain is shown in Figure 101, showing that whilst the load balance is not perfect, it is better than the balance that would have been obtained from a raw decomposition. The total number of elements used for the simulation was approximately 39 million tetrahedra and 730,000 vertices. The simulation result on the four aircraft is shown in Figure 102.

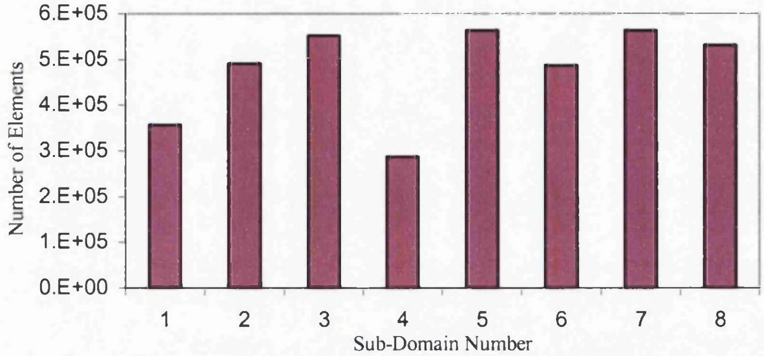


Figure 101 Work Load for 4 F15 Simulation

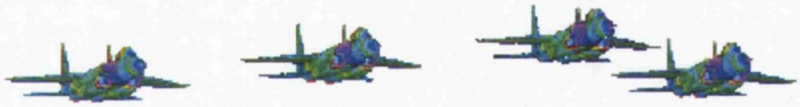


Figure 102 Four F15 Density Plots

In order to increase the complexity of the F15 geometry, in order to test the robustness and applicability of the parallel mesh generator with over-decomposition, the "Display" configuration was devised. Consisting of two F15 aircraft flying with the tail planes entwined, an inviscid calculation was performed at Mach 0.85 with incidence angle of five degrees. The work-load graph is shown in Figure 103, with the density plots from the solution rendered onto the aircraft shown in Figure 104. Again, the mesh was overdecomposed to 512 sub-domains, and assimilated to eight sub-domains.

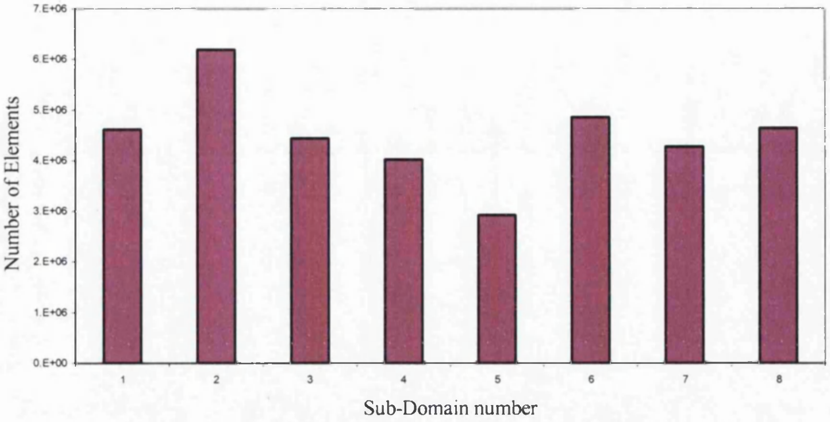


Figure 103 Work Load for 2 F15 Simulation

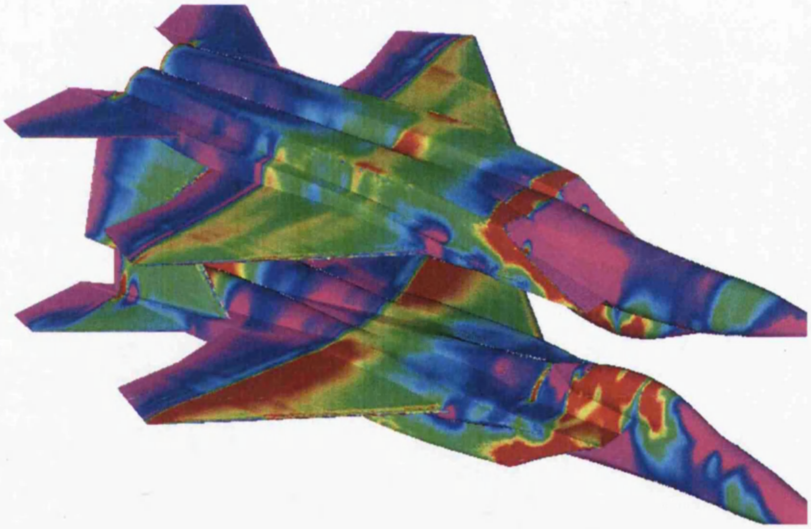


Figure 104 2 F15 Density Plots

5.1.3 F16 Military Aircraft Geometry

The F16 military aircraft geometry was used in full store configuration to determine speed up graphs for the parallel mesh generator. For flow feature resolution, the number of elements required to resolve the flow was approximately 60 million elements. Figure 105 shows a typical decomposition of the geometry, with cutting planes clearly shown. The aircraft is plotted in red, showing the large variation of element size from the immediate area around the aircraft to the outer boundary.

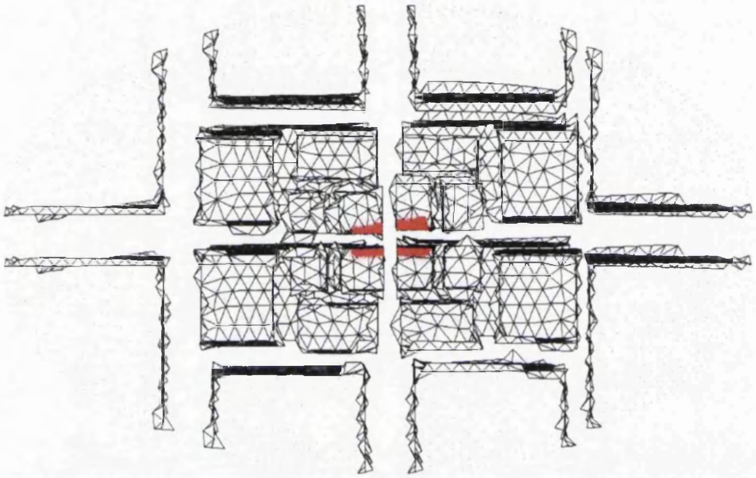


Figure 105 F16 32 sub-domain decomposition

The F16 geometry was chosen for timing tests in order to determine any speed-up observed as a by-product of parallelisation. Figure 106 shows the speed-up achieved by decomposing the domain into 32 sub-domains and assimilating to the number of processors for each particular case.

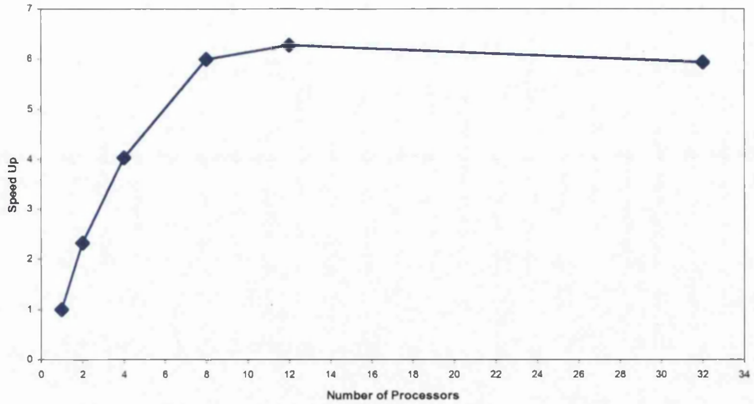


Figure 106 Speed up graph for F16 geometry

The graph shows that super-linear speed up is achieved between one and two processors and one and four processors. Increasing the number of processors beyond this, for this testcase, does not yield greater than a factor of 6 speed-up, although this is due in part to the reduction in load balance. Shown in Figure 107 to Figure 111 are the load balance graphs, showing that as the number of partitions to assimilate to approaches that of the maximum number of sub-domains to decompose the domain to before assimilation, the load balance of the assimilated domains reduces significantly. Figure 112 shows the volume partitioned mesh of the F16, where the edges of the sub-domain boundary have been plotted to show the complex nature of the sub-domains after assimilation.

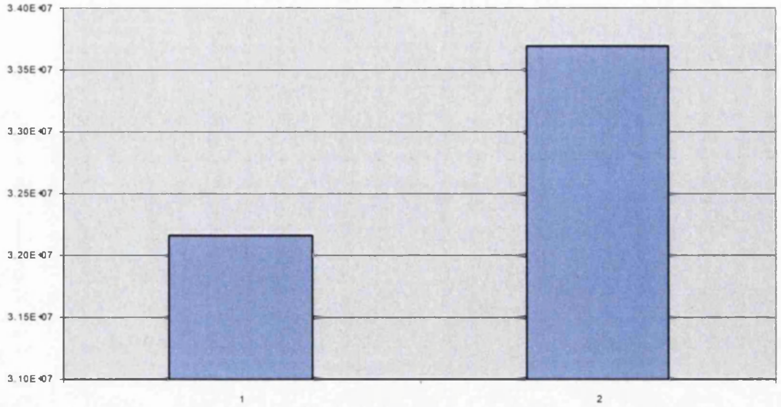


Figure 107 Load Balance (elements) for two sub-domain F16 mesh

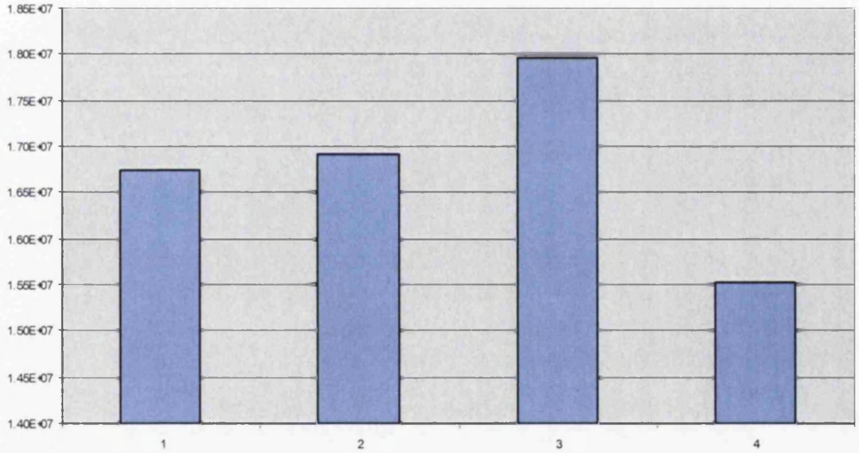


Figure 108 Load Balance (elements) for four Sub-domain F16 mesh

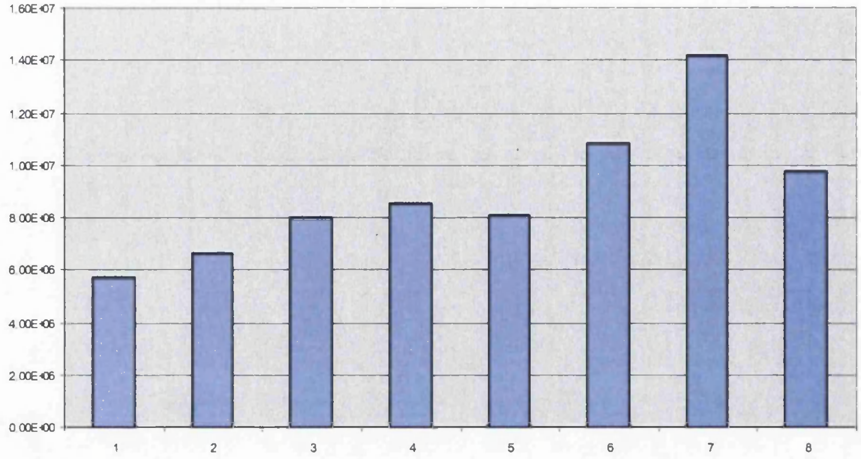


Figure 109 Load Balance (elements) for eight sub-domain F16 mesh

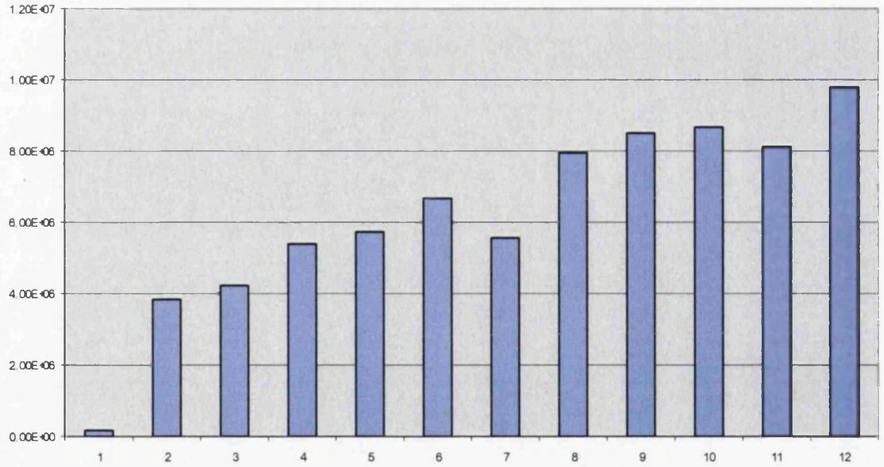


Figure 110 Load balance (elements) for twelve sub-domain F16 mesh

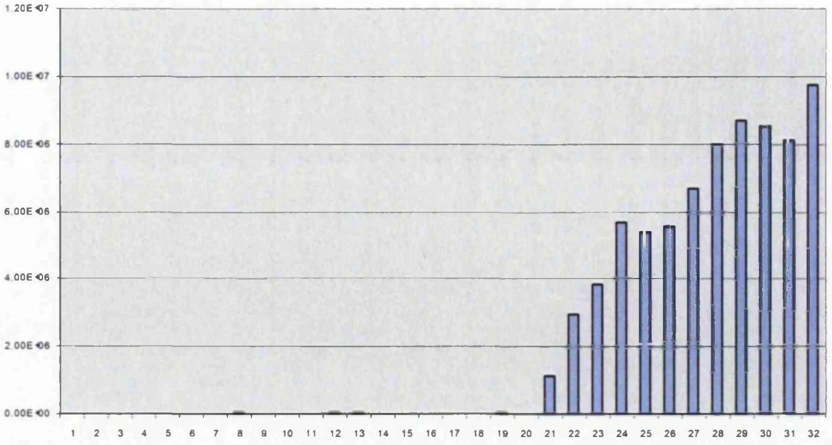


Figure 111 Load balance (elements) for thirty-two sub-domain F16 mesh

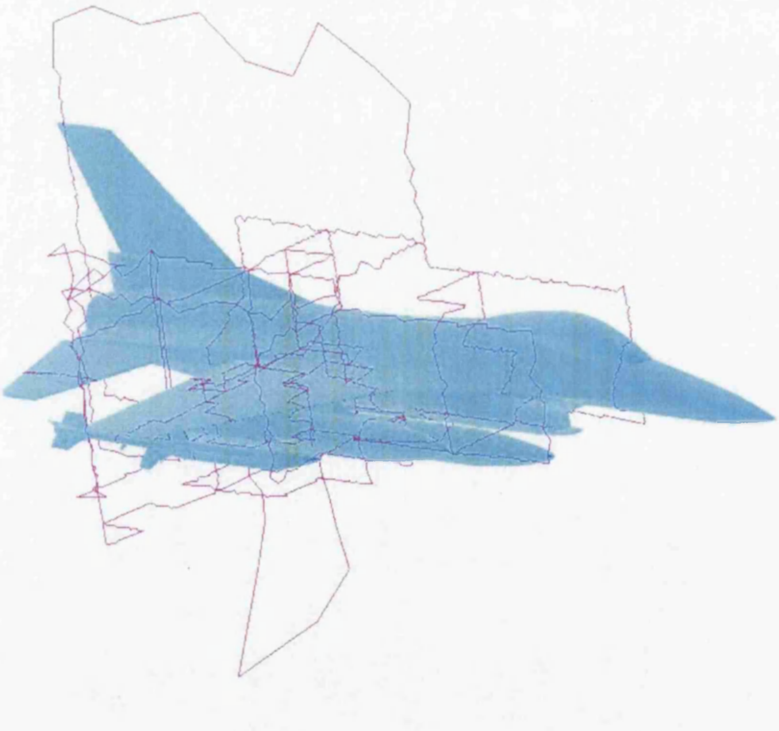


Figure 112 F16 assimilated mesh showing sub-domain edges

5.1.4 EADS Gulfstream Commercial Jet

In order to demonstrate the application of the boundary layer mesh generation scheme to a full aircraft geometry, the Gulfstream jet has been used. A boundary layer containing 15 layers of anisotropic elements was required. Generating the anisotropic region using the optimised advancing layer method showed memory usage of 50MB, compatred to 82MB for the sequential non-optimised generation scheme. In total, the maximum memory usage for the parallel mesh generator was 150MB, which compares favourably to the sequential generation method where 273MB of memory was required. Figure 113 shows a cut through the mesh, clearly showing the layers of anisotropic elements grown from the fuselage.

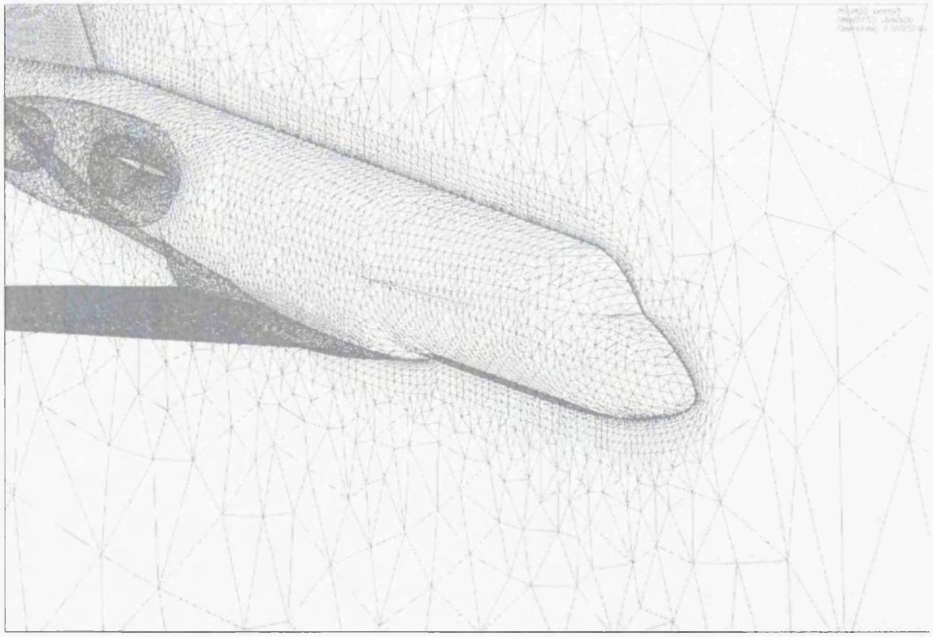


Figure 113 EADS Gulfstream anisotropic mesh

5.2 CEM Results

Electromagnetic scattering simulations place great demands on mesh generation programs. The use of low-order schemes to solve the linear Maxwell equations results in requirements for meshes with edge lengths that are proportional to the frequency of the simulation. Typical military and civil applications for electromagnetic scattering for aerospace applications require that the scattering wave has a frequency of greater than 8 GHz. The graph in Figure 98 shows the increase in mesh size for the scattering of an electromagnetic wave across an aircraft twenty metres in length. To perform these large simulation parallel computer architecture is required, and parallel meshing stands out as the only means of generating the large datasets, typically greater than 100 million tetrahedral elements, to enable the simulation to be performed. Three test cases are shown for the application of parallel meshing as a means to provide these large datasets. An aerospace engine duct, provided by BAESystems, shows the application to internal scattering problems with a simulation at 10 GHz. The scattering of an incident wave across the Dassault Falcon is shown for a simulation at 1 GHz, and a similar simulation of a trihedral cavity at 10 GHz.

5.2.1 Aerospace Engine Duct Simulation

The aerospace engine duct simulation represents a single incident wave entering the engine duct and reflecting back out. This type of scattering represents a problem for engineers designing stealth aircraft, and as such provides an opportunity to model sections of the aircraft individually. The simulation was to be performed at a frequency of 10 GHz, giving a wavelength of 3 centimetres. The shape of the duct was approximately a tube, of length 6.2 metres and diameter 0.45 metres. Assuming ten nodes per wavelength for the solver, gave a requirement of 47.25 million vertices, which is approximately 250 million elements. Due to the internal bounded nature of the simulation, a perfectly matched layer was not required. The uniformity of the electromagnetic mesh means that any over-decomposition is not required, and the load balance is good enough to perform the simulation on the raw decomposition result.

Figure 114 Duct Geometry

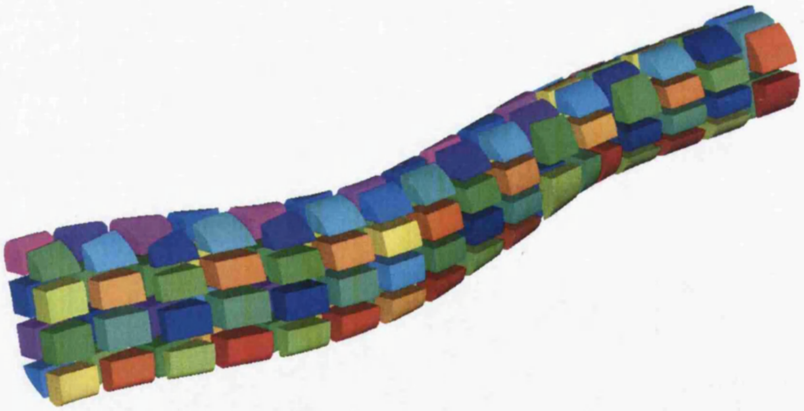


Figure 115 Duct Mesh showing partitions

The mesh was generated using 128 sub-domains, on 32 processors. This therefore required that dynamic load balancing be used; the number of sub-domains that each of the 32 processors generated is shown in Figure 117. The time that each processor was operating for is shown in Figure 116. This shows uniformity in the length that each processor took, although the number of sub-domains generated varies. It should be noted that the programme was run in full debugging mode to generate the sub-domains and volume meshes, which accounts for the length of time taken.

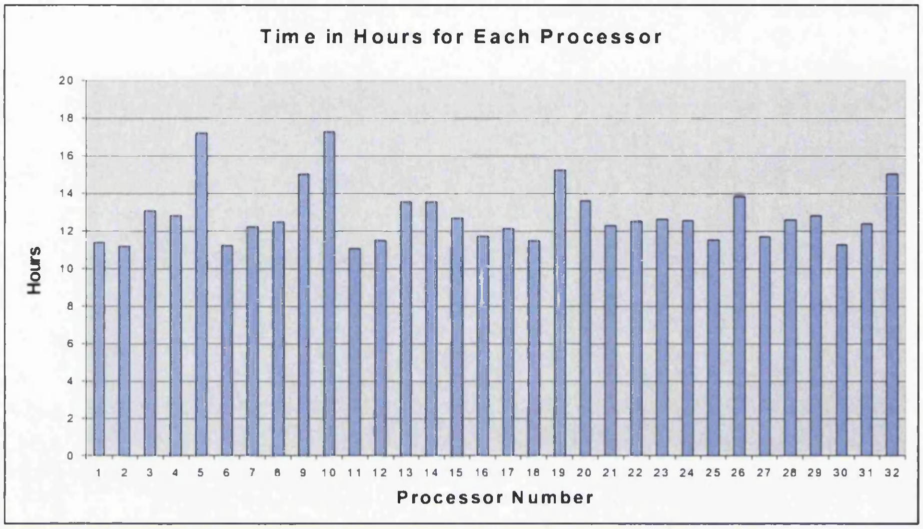


Figure 116 Timings for each processor for duct generation

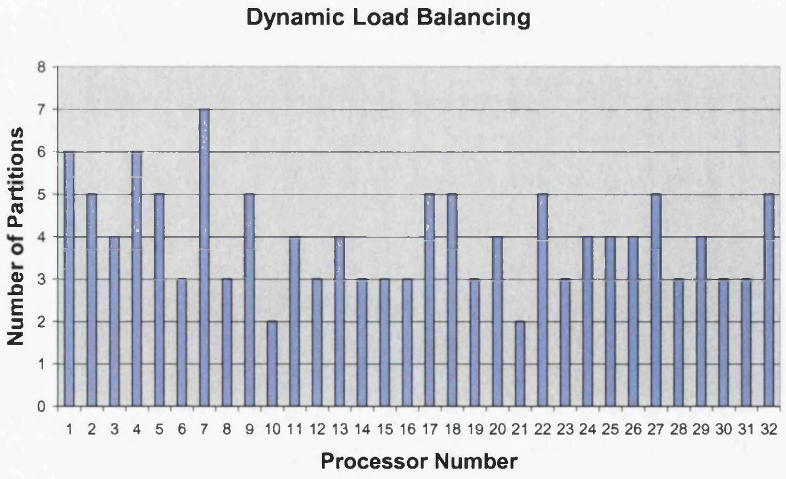


Figure 117 Dynamic Load Balancing for each processor

5.2.2 Dassault Falcon Simulation

In order to perform the simulation of a single incident wave across a commercial jet, here the Dassault Falcon, a large number of elements are required. The simulation was performed at a frequency of 1 GHz, requiring the domain be discretised with 500 million elements. The mesh statistics are shown in Figure 118, and the simulation after one cycle in Figure 122. The load balance, excluding PML region is shown in Figure 119.

	Number of Vertices	**Number of Elements**
Surface Mesh	$1.4\text{x}10^6$	$4.2\text{x}10^6$
Volume Mesh	$498.7\text{x}10^6$	$10.4\text{x}10^6$

Figure 118 Mesh statistics for Dassault Falcon simulation

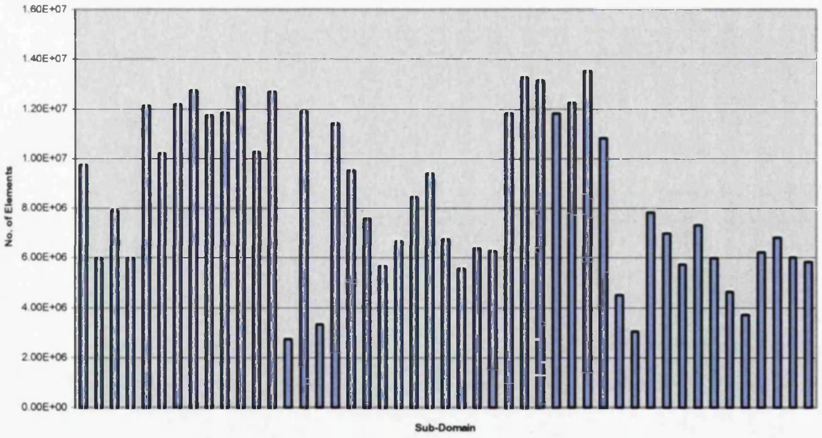


Figure 119 Load balance for 500 million element Falcon simulation

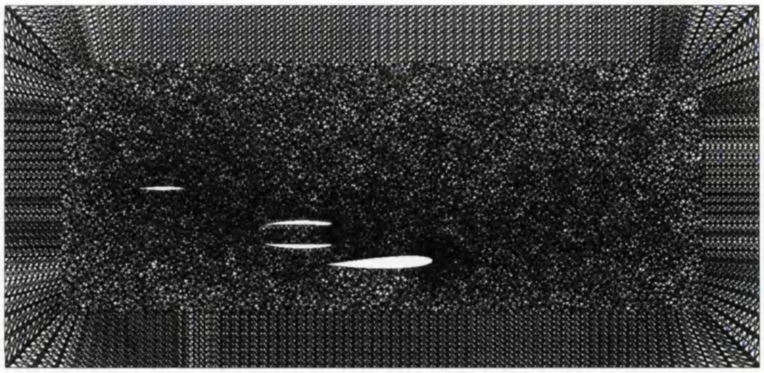


Figure 120 Falcon CEM Mesh, showing PML region and isotropic region

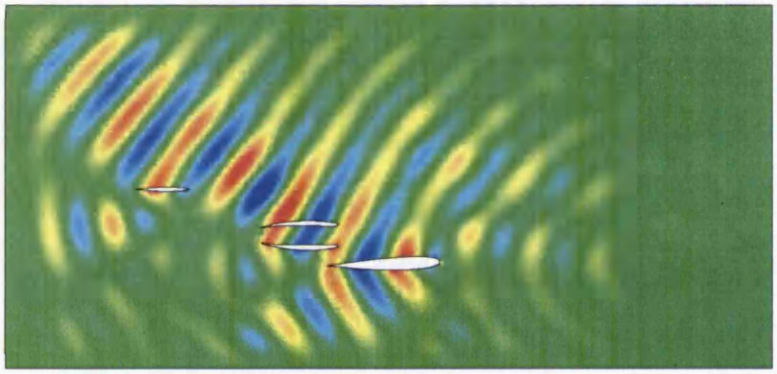


Figure 121 Scattering cross section of Falcon simulation showing engine nacelle and tail plane cuts

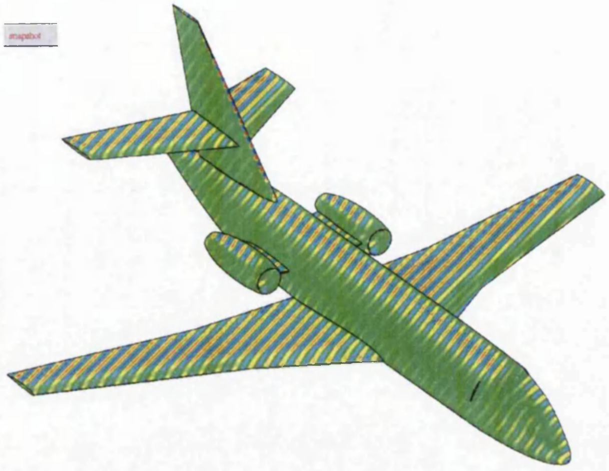


Figure 122 Solution after a single cycle for Dassault Falcon

5.2.3 Trihedral Cavity Simulation

A number of simulations were performed in conjunction with the University of Minnesota to compare the scattering of a single incident electromagnetic wave into a trihedral cavity. The geometry is shown in Figure 124, and the solution at 20 GHz is shown in Figure 125. A number of frequencies were simulated, finishing in a simulation of 20 GHz. This frequency required a mesh size of approximately 1 billion tetrahedral elements, including PML region containing approximately 100 million elements. The mesh sizes generated are shown in Figure 123.

	Number of Vertices	**Number of Elements**
Surface Mesh	$1.3\text{x}10^6$	$4.1\text{x}10^6$
Volume Mesh	$183\text{x}10^6$	$980\text{x}10^6$

Figure 123 Mesh sizes for trihedral cavity simulation

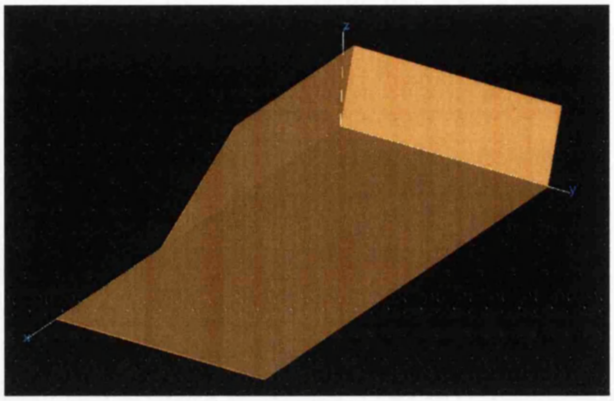


Figure 124 Trihedral Cavity Geometry

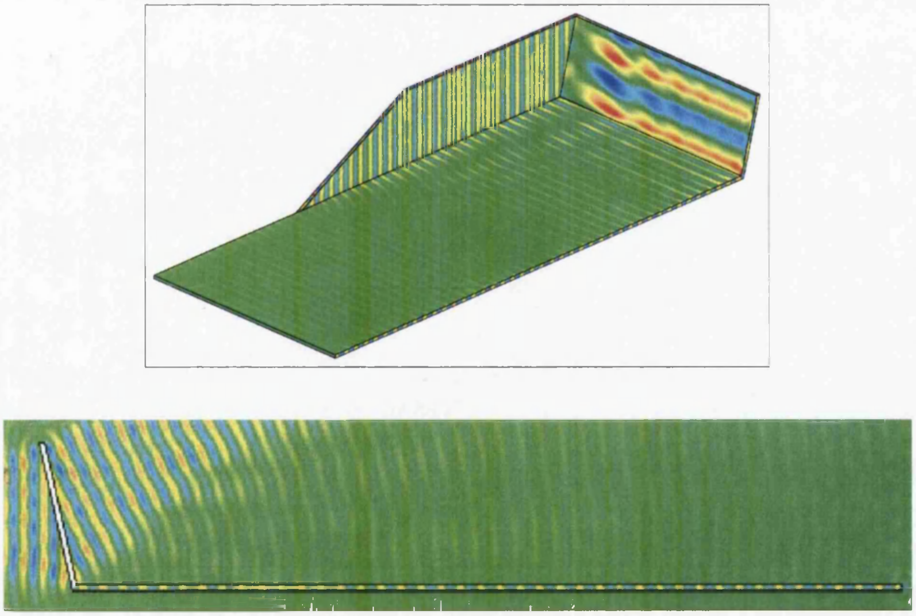


Figure 125 Electromagnetic Scattering solution plotted on geometry

Generated on an SGI Origin class parallel computer, the surface mesh was subdivided into 63 isotropic sub-domains, with a final sub-domain used to contain the PML region. Each isotropic sub-domain contains approximately 15 million tetrahedral elements. The sub-domain element load distribution is shown in Figure 127.

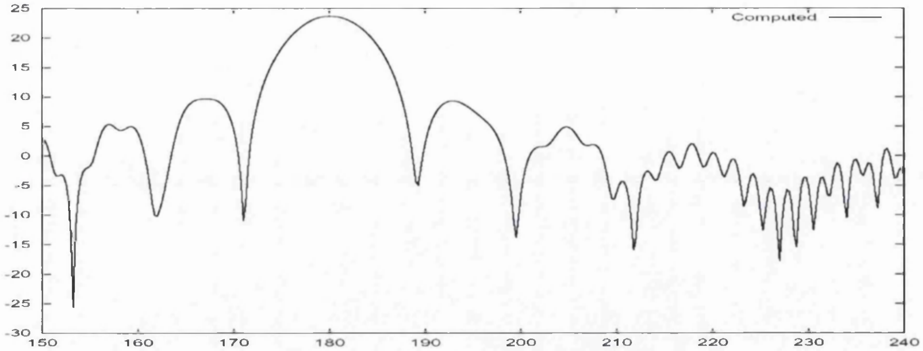


Figure 126 RCS for Trihedral Simulation

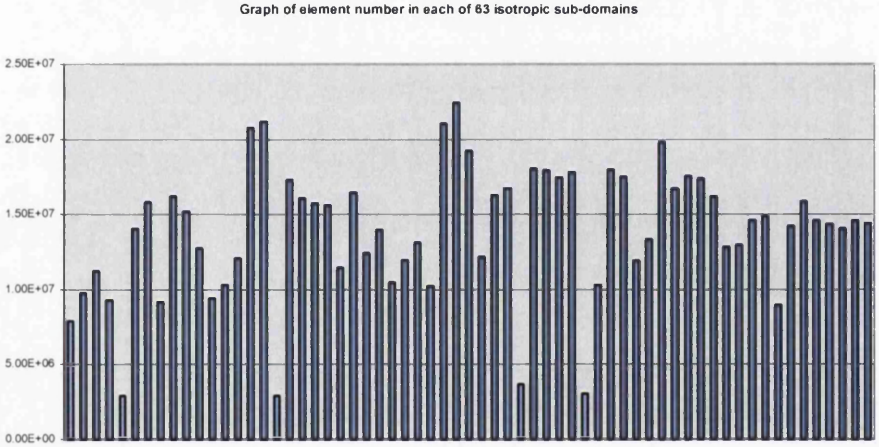


Figure 127 Element load for Trihedral cavity simulation

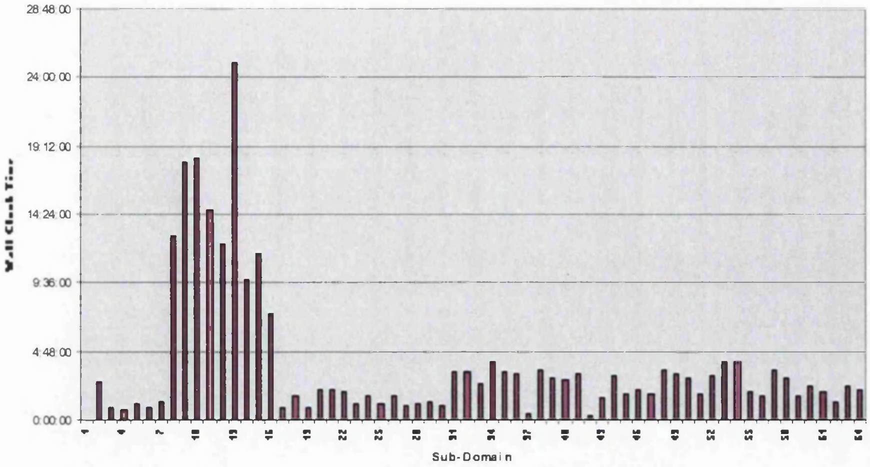


Figure 128 Graph of wall clock time for each sub-domain of trihedral cavity simulation

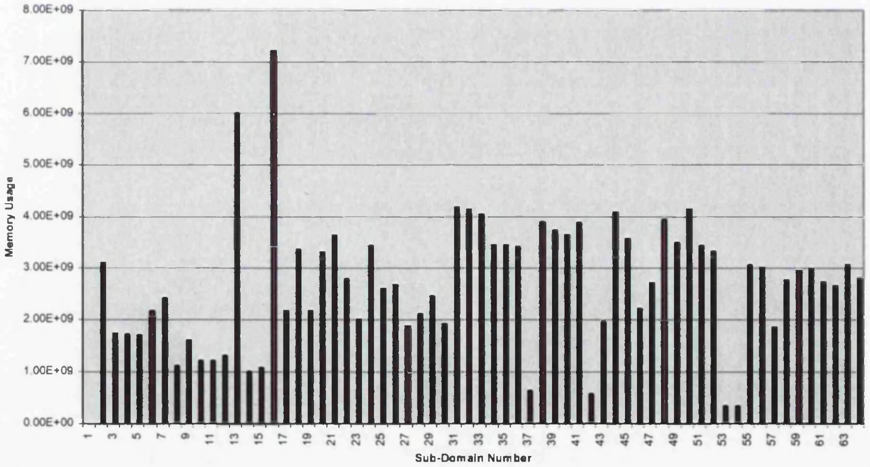


Figure 129 Graph of memory usage for each sub-domain in trihedral cavity simulation

6 Conclusion

The work presented in this thesis has covered aspects of mesh generation related to the simulation of fluid flow and electromagnetic scattering. Initially, the two types of mesh were introduced, structured and unstructured. The methods developed to generate these types of meshes have been described.

Section 3 introduced the Delaunay method for creating a triangulation of a set of points and the various implementa for creating Delaunay satisfying meshes. The use of anisotropic or stretched elements was explained in terms of computational efficiency and solution accuracy for fluid flow simulations. The extension of the Delaunay method to use ellipses in place of circles in order to generate these stretched elements was shown.

A new method to specify the metrics that describe the ellipses was developed as an extension of the source method for point density control. Shown with examples of its application for two-dimensional meshes, the directional sources can be applied where a-priori knowledge of the flow directions is known. This would allow the developer to generate initial meshes containing stretched elements, such that the simulations would require less points than for an isotropic mesh for similar flow solution resolution. This mesh could then form the basis for an adaptive simulation, where cycles of solution and mesh adaptation are used to resolve flow solution and reduce error across the domain. An adaptive scheme to adapt a mesh to contain anisotropic elements has also been introduced. By determining the metric map that defines the ellipses from a previous solution, the mesh can be modified accordingly to conform to this metric map. The scheme ensures that the mesh is Delaunay satisfying with respect to the metric map, and contains directional point density where dictated by the solution. Obtaining the metric map and adapting the mesh provides a measure of the error across the domain. Problems of the accuracy of the metric map with respect to the size of ellipses were discussed, and a method to restrict the cavity to neighbouring elements suggested. Results have been shown that demonstrate this restriction, and how the solution improved by using this method. The cavity modification ensures that the structure of the mesh follows the solution, and that the mesh quality indicators, such as number of elements at a point. do not exceed recommended values.

Section 4 introduced the use of parallel computing for mesh generation tasks. The reasons behind the use of parallel computing in simulations of fluid and electromagnetic problems were discussed. A method for generating isotropic meshes was introduced initially. The various problems associated with the generation of meshes in parallel (for example load balancing and final mesh quality) were discussed, and methods to alleviate these problems described. A method to balance the volume mesh generation task, by over-decomposing the mesh and assimilating to fewer tasks, results in sub-domains that have improved load balance compared to that of the initial sub-domains provided by the divide and conquer scheme. Results for these type of meshes for geometries varying in complexity from a simple M6 wing to an aircraft in full store configuration were shown. In order to challenge both the flow solver and the mesh generator, the meshes for four entire aircraft in 'finger of four' configuration and two aircraft in an unrealistic configuration were generated. The flow solutions for these simulations were shown. The parallel mesh generator has also been extended to provide boundary layer definition capability. By optimising the advancing layer technique such that only a single layer of elements is held in memory at once, the memory overhead requirement is kept to a minimum. The meshes were validated on a number of testcases shown in Section 5. In addition to the boundary layer definition, the advancing layer technique was used to discretise the perfectly matched layer region for electromagnetic scattering simulations. This technique was applied to three testcases; a PEC sphere, a full aircraft geometry and a trihedral cavity. The simulations ranged in size from 100 million elements to 1 billion elements.

Further work for the anisotropic Delaunay work would be to extend the principles to three-dimensions. Problems have been reported [41] that can reduce the applicability and robustness of the method. In two dimensions the use of anisotropic elements to reduce computational expense is not required, due to the relative inexpense of computer hardware and because of the size of problems tackled. In three-dimensions, an adaptive mesh scheme could be used, where an initial mesh containing anisotropic elements which have been defined by directional point sources is used as the starting point. From an initial solution on this mesh, a metric map can be obtained for the adaptation loop to begin. The savings of anisotropic elements are only noticed in three-dimensions, where the mesh size can rapidly grow beyond the memory capabilities of most workstations. The use of the cavity restriction would also aid in creating mesh structure in the regions of adaptation.

A parallel mesh generator has been created that is both stable and memory efficient. The program is used in industry to generate meshes in aerospace research institutes across Europe. The results shown range from meshes of a few thousand elements, where the parallel aspects are not required, to meshes containing hundreds of million elements. The bottleneck of simulations has traditionally been the mesh generator, since flow solvers have been relatively scalable with mesh size, reliant on simply porting to a larger machine in terms of processors or memory. The use of graph partitioning libraries to partition the meshes have allowed meshes to be distributed to parallel machines whilst balancing workload and minimising communication. By using the parallel scheme meshes can be generated that allow the simulation of physical problems previously not possible with current linear solver technology. In the future, higher order methods may be used in order to reduce the number of elements required to resolve the solution. Whilst generating meshes in size of up to 500 million elements has been possible for fluid flow and electromagnetic scattering simulations in an automatic way, generating very large meshes containing over 750 million elements begins to test the software to the point where the user must intervene.

APPENDIX A

A bivariate quadratic curve centred on the origin has the equation:

$$ax^2 + 2bxy + cy^2 = 0$$

$$J = \begin{vmatrix} a & b \\ b & c \end{vmatrix}$$

If J>0 then this equation defines an ellipse centred on the origin. Parameters a and c control the stretching along the major and minor axes, and b controls the rotation of the minor and major axes. Therefore, b can be replaced by a rotation:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos\vartheta & \sin\vartheta \\ -\sin\vartheta & \cos\vartheta \end{bmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix}$$

So

$$\begin{aligned}x &= x^{'}\cos\vartheta + y^{'}\sin\vartheta\\ y &= -x^{'}\sin\vartheta + y^{'}\cos\vartheta\end{aligned}$$

Substituting these equations we get:

$$\begin{aligned}x^{2} &= (x^{'}\cos\vartheta + y^{'}\sin\vartheta)(x^{'}\cos\vartheta + y^{'}\sin\vartheta)\\ &= x^{'2}\cos^{2}\vartheta + y^{'2}\sin^{2}\vartheta + 2x^{'}y^{'}\sin\vartheta\cos\vartheta\end{aligned}$$

$$\begin{aligned}y^{2} &= (-x^{'}\sin\vartheta + y^{'}\cos\vartheta)(-x^{'}\sin\vartheta + y^{'}\cos\vartheta)\\ &= x^{'2}\sin^{2}\vartheta + y^{'2}\cos^{2}\vartheta - x^{'}y^{'}\sin\vartheta\cos\vartheta - y^{'}x^{'}\cos\vartheta\sin\vartheta\end{aligned}$$

$$\begin{aligned}xy &= (x^{'}\cos\vartheta + y^{'}\sin\vartheta)(-x^{'}\sin\vartheta + y^{'}\cos\vartheta)\\ &= y^{'2}\sin\vartheta\cos\vartheta + x^{'}y^{'}(\cos^{2}\vartheta - \sin^{2}\vartheta) - x^{'2}\cos\vartheta\sin\vartheta\end{aligned}$$

Substituting these equations into (1) gives:

$$\begin{aligned}&ax^{'2}\cos^{2}\vartheta + ay^{'2}\sin^{2}\vartheta\\ &+ 2by^{'2}\sin\vartheta\cos\vartheta + 2bx^{'}y^{'}(\cos^{2}\vartheta - \sin^{2}\vartheta) - 2bx^{'2}\cos\vartheta\sin\vartheta\\ &+ cx^{'2}\sin^{2}\vartheta - 2cx^{'}y^{'}\sin\vartheta\cos\vartheta + cy^{'2}\cos^{2}\vartheta = 0\end{aligned}$$

Collecting terms gives us:

$$\begin{aligned}&x^{'2}(a\cos^{2}\vartheta - 2b\cos\vartheta\sin\vartheta + c\sin^{2}\vartheta) +\\ &y^{'2}(a\sin^{2}\vartheta + 2b\sin\vartheta\cos\vartheta + c\cos^{2}\vartheta) +\\ &x^{'}y^{'}(2a\sin\vartheta\cos\vartheta + 2b(\cos^{2}\vartheta - \sin^{2}\vartheta) - 2c\sin\vartheta\cos\vartheta) = 0\end{aligned}$$

Comparing coefficients

$$
\begin{aligned}
&a'x'^2 + 2b'x'y' + c'y'^2 = 0 \\
&a' = a\cos^2\vartheta - 2b\cos\vartheta\sin\vartheta + c\sin^2\vartheta \\
&b' = a\sin\vartheta\cos\vartheta + b\cos^2\vartheta - b\sin^2\vartheta - c\sin\vartheta\cos\vartheta \\
&c' = a\sin^2\vartheta + 2b\sin\vartheta\cos\vartheta + c\cos^2\vartheta
\end{aligned}
$$

Hence in order to make cross terms vanish, need to set

$$
\begin{aligned}
&b' = 0 \\
&b' = b(\cos^2\vartheta - \sin^2\vartheta) - (c-a)\sin\vartheta\cos\vartheta \\
&= b\cos(2\vartheta) - \tfrac{1}{2}(c-a)\sin(2\vartheta) = 0
\end{aligned}
$$

For this to be true:

$$
\begin{aligned}
&b\cos(2\vartheta) = \tfrac{1}{2}(c-a)\sin(2\vartheta) \\
&\frac{\cos(2\vartheta)}{\sin(2\vartheta)} = \frac{(c-a)}{2b} = \cot(2\vartheta)
\end{aligned}
$$

So, given a stretching along the principal axes, a and c, and the required rotation, θ, it is possible to determine b using the formula:

$$
2b = \frac{(c-a)}{\cot(2\vartheta)} \Rightarrow b = \frac{(c-a)}{2\cot(2\vartheta)}
$$

REFERENCES

1 Zienkiewicz OC, Taylor RJ. *The Finite Element Method, Volume 1 The Basis (fifth edition)*; Butterworth and Heinemann 2000.

2 Versteeg HK, Malalaseker A; *An introduction to computational fluid dynamics*; Prentice-Hall 1995

3 http://www.catia.com

4 http://www.ptc.com

5 Thompson JF, Warsi ZUA, Mastin CW; *Numerical grid generation, foundation and applications*; North Holland 1985

6 Gordon WJ, Hall CA; *Construction of curvilinear coordinate system and applications to mesh generation*; International Journal For Numerical Methods in Engineering;1973

7 Frey PJ, George P-L; *Mesh generation application to finite elements*; hermes-science Paris 2000

8 Thompson JF, Soni BK, Weatherill NP (Editors); *Handbook of Grid Generation*; CRC Press New York 1999

9 George A; *Computer implementation of the finite element method*; PhD Thesis Stanford University 1971

10 Bowyer A; *Computing Dirichlet tessellations*; The Computer Journal 1981; Vol 24 no 2: pp 162-166.

11 Watson DF; *Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes*; The Computer Journal 1981; Vol 24 no 2: pp 167-172.

12 Yerry M, Shephard MS; *Automatic Three-Dimensional Mesh Generation By The Modified Octree Technique*; International Journal For Numerical Methods in Engineering 1984; Vol 20 pp.1965-1990

13 Owen SJ; *Meshing software survey*; Published online http://www.andrew.cmu.edu/user/sowen/softsurv.html

14 Schneiders R; *A grid-based algorithm for the generation of hexahedral element meshes*; Engineering with Computers 1996; Vol 12 pp168-177

15 Tautges TJ, Blacker T, Mitchell SA; *The Whisker Weaving Algorithm: A Connectivity-Based Method for Constructing All-Hexahedral Finite Element Meshes*; International Journal for Numerical Methods in Engineering 1996;Vol 39 pp.3327-3349

16 Sibson R; *Locally Equiangular triangulations*; The Computer Journal;1978 Vol 21 pp243-245

17 Ruppert J; A Delaunay *Refinement Algorithm for Quality 2-Dimensional Mesh Generation*; Journal of Algorithms, pp.1-45, Feb 1994

18 Weatherill NP, Hassan O; *Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints*; International Journal for Numerical Methods in Engineering *1994*; Vol 37 pp 2005-2039

19 Löhner R, Morgan K. *Improved adaptive refinement strategies for finite element aerodynamic computations*; From proceedings of American Institute of Aeronautics and Astronautics, 24th Aerospace Sciences Meeting, January 6-9 1986. AIAA-86-0499.

20 Löhner R, Cerbral J. *Generation of non-isotropic unstructured meshes via directional enrichment;* International Journal for Numerical Methods in Engineering 2000; Vol 49: pp219-232.

21 Mavriplis DJ; *Unstructured and adaptive mesh generation for high Reynolds number viscous flow*; Numerical Grid Generation in Computational Fluid Mechanics, Pineridge Press; 1988 pp 611-620

22 Peraire J, Vahdati M, Morgan K, Zienkiewicz OC. *Adaptive remeshing for compressible flow computations.* Internal report CR/R/544/86, Institute for Numerical Methods in Engineering, University College, Swansea. 1986.

23 Vallet M-G; *Generation de maillages element finis anistropes et adaptifs*; These de l'Universite Paris 6 1992.

24 Borouchaki H, Castro-Díaz MJ, George PL, Hecht F, Mohammadi B; *Anistropic adaptive mesh generation in two dimensions for CFD*; From 5th International Conference On Numerical Grid Generation in Computational Field Simulations, Mississippi State University, Vol 3, pp.197-206, April 1996

25 Castro-Díaz MJ, Hecht F, Mohammadi B; *New progress in anisotropic mesh adaptation for inviscid and viscous flows simulations*; INRIA Report No. 2671, INRIA, October 1995.

26 Tam A, Ait-Ali-Yahia D, Robichaud MP, Moore M, Kozel V, Habashi WG. *Anisotropic mesh adaptation for 3D flows on structured and unstructured meshes*; Computer methods in applied mechanical engineering. 2000; Vol 189: pp1205-1230.

27 Borouchaki H, George PL, Hecht F, Laug P, Saltel E; *Delaunay mesh generation governed by metric specifications, part I Algorithms*; Finite Elements in Analysis and Design. 1997; Vol 25: pp 61-83.

28 Borouchaki H, George PL, Mohammadi B; *Delaunay mesh generation governed by metric specifications, part II Applications;* Finite Elements in Analysis and Design. 1997; Vol 25: pp 85-109.

29 Pain CC, Umpleby AP, de Oliveira CRE, Goddard AJH; *Tetrahedral mesh optimisation and adaptivity for steady-state and transient finite element calculations;* Computer Methods in Applied Mechanical Engineering. 2001; Vol 190:pp3771-3796.

30 Castro-Díaz MJ, Hecht F, Mohammadi B, Pironneau O; *Anisotropic unstructured mesh adaptation for flow simulations.* International Journal for Numerical Methods in Fluids. 1997; Vol 25: pp475-491.

31 Borouchaki H, Frey PJ. *Adaptive triangular–quadrilateral mesh generation.* International Journal for Numerical Methods in Engineering. 1998; Vol 41: pp915-934.

32 Castro-Díaz MJ, Hecht F. *Anisotropic Surface Mesh Generation.* INRIA Rapport de recherché; October 1995; Number 2672.

33 Yamada A, Shimada K, Itoh T. *Meshing curved wire-frame models through energy minimization and packing of ellipses.* International Journal for Numerical Methods in Engineering; 1999; Vol 46:pp 1221-1236.

34 Lee CK. *Automatic metric advancing front triangulation over curved surfaces.* Engineering Computations; Vol 17:pp48-74.

35 Lee YK, Lee CK. *Automatic generation of anistropic quadrilateral meshes on three-dimensional surfaces using metric specification.* International Journal for Numerical Methods in Engineering; 2002; Vol 53: pp2673-2700.

36 Dirichlet GL. *Uber die Reduction der positiven Quadratischen formen met drei Enderstimmten Ganzen Zahlen; Z. Reine Angew, Mathematics* 1850; Vol 40 no 3: pp209-227.

37 Voronoi G; *Nouvelles applications des parameters continues a la theore des formes quadratiques.* Recherches sur les parallelloedres primitfs, Journal Reine Angew. Mathematics 1908; Vol 134.

38 Delaunay B. *Sue la sphere vide.* Bulletin of Academic Science URSS. Science National 1934: pp 793-800.

39 Green PJ, Sibson R; *Computing Dirichlet tessellation in the plane.* The Computer Journal 1981; Vol 21 no 2: pp 168-173.

40 Weatherill N. P.; *The reconstruction of boundary contours and surfaces in arbitrary unstructured triangular and tetrahedral grids*; Engineering Computations; Vol. 13, No.8, 1996, pp66-81.

41 George PL, Borouchaki H. *Anisotropic Delaunay based mesh generation.* Von Karman Institute for Fluid Dynamics Lecture Series, 31st Computational Fluid Dynamics, 20-24 March 2000.

42 Moore GE; *Cramming more components onto integrated circuits*; Electronics 1965; Vol 38 (8)

43 Amdahl GM; *Validity of single-processor approach to achieving large scale computing scalability*; Proceedings of AFIPS Conference Reston, VA 1967;pp483-485

44 Gustafson JL; *Re-evaluating Amdahl's law*; CACM 1988;Vol 31(5) pp532-533

45 Kumar V; Introduction to parallel computing; design and analysis of parallel algorithms;Benjamin/Cummings Publishers, California 1994

46 CONDOR : http://www.cs.wisc.edu/condor

47 LSF : http://www.platform.com

48 Aliabadi S, Zellas J, Ahedi J, Johnson A, Berger C, Smith J; *Implicit, large-scale, parallel 3D simulation of waves impacting floating vessels*; AHPCRC Technical Report 1002-102, Minneapolis

49 de Cougny HL, Shephard MS. *Parallel Volume Meshing using face removals and hierarchical repartitioning*. Computer Methods in Applied Mechanical Engineering; 1999; Vol 174: pp275-298.

50 Lohner R, Camberos J, Merriam M; *Parallel Unstructured Mesh Generation*. Computer methods in applied mechanics and engineering; 1992; Vol 95: pp343-357.

51 Gaither A, Marcum D, Reese D, Weatherill NP; *A Paradigm for Parallel Unstructured Grid Generation*; 5th International Conference on Numerical Grid Generation in Computational Field Simmulations; Mississippi State University, pp.731-740, April 1996

52 Simon H; *Partitioning of unstructured problems for parallel processing*; Computer Systems in Engineering 1991; Vol2 pp135-148

53 Lammer L, Burghardt M; *Parallel generation of triangular and quadrilateral meshes*; Advances in Engineering Software; 2000(31) 929-936

54 Topping BHV, Cheng B; *Parallel and distributed adaptive quadrilateral mesh generation*; Computers and Structures; 1999 (73) 519-536

55 Lohner R; *Parallel Advancing Front Mesh Generation Scheme*;

56 Coupez T, Digonnet H, Ducloux R; *Parallel meshing and remeshing*; Applied Mathematical Modelling; 2000 (25):153-175

57 Rypl D, Bittnar Z; *Parallel 3D Mesh Generator*

58 Said R, Weatherill NP, Hassan O, Morgan K, Verhoeven NA; *Distributed parallel Delaunay mesh generation*; Computer Method in Applied Mechanical Engineering 1999; vol 177 pp109-125

59 Said R, Larwood BG, Weatherill NP, Hassan O, Morgan K; *Parallel Delaunay unstructured mesh generation*; Proceedings of the 7th International Conference on Numerical Grid Generation in Computational Field Simulation 2000;

60 Okunsaya T, Peraire J; *3D Parallel Unstructured Mesh Generation;* Trends in Unstructured Mesh Generation; AMD-Volume 220; ASME 1997

61 Chew LP, Chrisochoides N, Sukup F; *Parallel Constrained Delaunay Meshing;* Trends in Unstructured Mesh Generation; AMD-Volume 220; ASME 1997

62 Chrisochoides N, Nave D; *Parallel Delaunay Mesh Generation Kernel*; International Journal for Numerical Methods in Engineering 2002;

63 de Cougny HL, Shephard MS, Ozturan C; *Parallel Three-Dimensional Mesh Generation on Distributed Memory MIMD Computers*; Engineering with Computers; 1996 (12) 94-106

64 Wu P, Houstis EN; *Parallel Adaptive Mesh Generation and Decomposition*; Engineering with Computers; 1996(12) 155-167

65 Gropp W, Lusk E, Doss N, Skjellum A; *A high-performance, portable implementation of the (MPI) message passing interface standard*; Parallel Computing; 1996(22) 789-828

66 Geist A, Beguelin A, Dongeurra J, Jiang W, Mandel R; *PVM: Parallel Virtual Machine, A users guide and tutorial for Networked Parallel Computing*; 1994 MIT Press

67 Verhoeven NA, Jones J, Weatherill NP, Morgan K; *PVM and MPI applied to a master/slave parallel mesh generator*; PPECC Workshop 1995

68 Farhat C, Lesoinne M; *Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics*; International Journal for Numerical Methods in Engineering; 1993(36) 745-764

69 Karypis G, Schloegel K, Kumar V; *Parallel Graph Partitioning and Sparse Matrix Ordering Library*; University of Minnesota 1998

70 Hendrickson B, Leland R; *The CHACO users guide*; Sandia National Labs, Albuquerque 1995

71 Walshaw, C; *Parallel Jostle Library Interface*; University of Greenwich 2000

72 Hendrickson B, Kolda TG; Graph partitioning models for parallel computing; Parallel Computing 2000; Vol 26 pp 1519-1534

73 Jones JW; *An investigation into Visualisation for Computational Simulation*; PhD Thesis, Swansea, 2003

74 ICEM CFD : http://www.icemcfd.com

75 TGRID : http://www.fluent.com

76 Pirzadeh S; *Viscous Unstructured three-dimensional grids by the Advancing layers method*; 32nd Aerosapce Sciences Meeting and Exhibit 1994; AIAA-94-0417

77 Collino F, Monk PB; *Optimizing the perfectly matched layer*; Computer Methods for Applied Mechanical Engineering 1998; Vol 164 pp157-171

78 Berenger JP; *A perfectly matched layer for the absorption of electromagnetic waves*; Journal of Computational Physics 1994; Vol 114 pp185-200