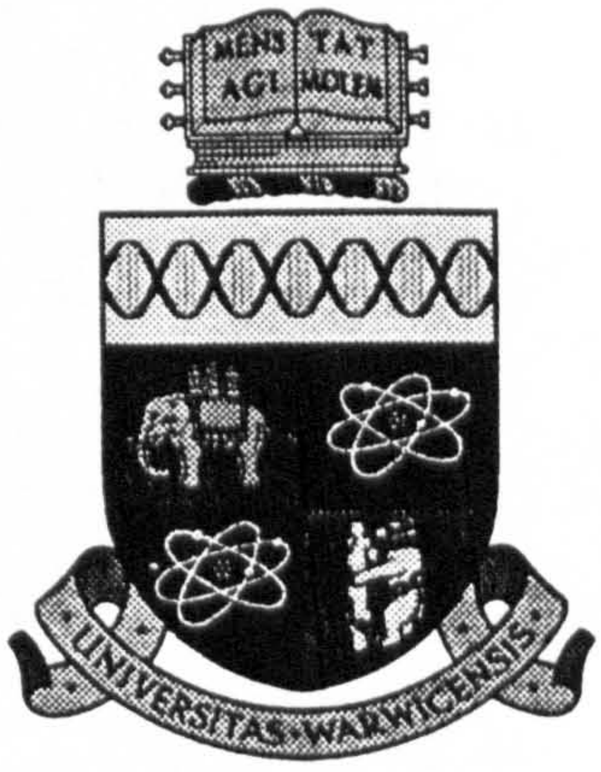
Using CFD in Engine Design

By

J L Graysmith

A thesis submitted for the Degree of

Doctor of Philosophy



Department of Engineering

University of Warwick

April 1995

ACKNOWLEDGEMENTS

I would like to thank Jaguar Cars Ltd and MIRA for funding this doctoral programme. In particular my thanks go to Steve Richardson, Steve Barraclough and Juliet McCleod at Jaguar who were both supportive of this work and excellent colleagues to work with. Also, this thesis could not have been completed without the unreserved support of Paul Atkin and Alan Draper at MIRA for which I am very grateful.

My greatest thanks go to my supervisor, Chris Shaw, whose efforts have helped me throughout this work and from whom I have learned so much. In addition, I would like to thank everyone in the Fluid Dynamics research 'team' at Warwick University for their detailed discussions and moral support.

I would like to dedicate this thesis to my partner, Chris Halliday, who has suffered for my art most of all.

DECLARATION

Parts of the work presented in this thesis have been published in

C.T Shaw, J.L. Graysmith, S.H. Richardson.

Influencing engine design using computational fluid dynamics.

SAE Paper 930877, 1993.

J.L. Graysmith, C.T Shaw.

Reducing Model Preparation Times for Industrial CFD Problems.

Numerical Methods in Laminar and Turbulent Flow 1993 Vol. 8 Part 2

p1370-1381. Ed. C. Taylor

SUMMARY

In this thesis the author presents two areas of work; exploring the integration of Computational Fluid Dynamics (CFD) into engine design for Jaguar Cars Ltd and developing a novel 'mesh construction' method for making mesh generation both easy and fast.

It is concluded that Jaguar can use CFD in the evaluation stage of the engine design process, although not in the concept stage of design. The CFD predictions are shown to be useful for detecting flow related faults and determining the general flow trends, but they should not be used as an absolute measure of the flow variables. The author has determined an efficient method for obtaining good quality meshes using commercial modelling and mesh generation software which requires a skilled CFD analyst. Steady flow analysis of an engine port and cylinder design could currently be completed in about six weeks using a high-powered workstation. The author recommends dedicated workstations for CFD analysis and training Jaguar's draughtsmen to create CAD models with computer analysis requirements in mind.

The author's mesh construction program automatically joins two overlapping meshes or cuts one mesh from another. Whilst the program works well on the test cases considered, it is not at a stage for commercial exploitation. Further development is therefore recommended.

TABLE OF CONTENTS

List of Figures

Chapter 3

Chapter 4

LIST OF TABLES

1. INTRODUCTION

1.1 Practical relevance

In internal combustion (IC) engines, the motion of the air and fuel mixture within the combustion chamber has a strong influence upon the combustion process, as described by Heywood [1]. Combustion, however, affects the exhaust emissions, efficiency and power output of an engine. Therefore, determining how the fluid motion affects an engine's performance could allow engineers to design engines with optimal flow fields for combustion and so create better engines.

Physical experiments have been used extensively to measure and visualise both the flow field and combustion. Computer experiments can also be performed using Computational Fluid Dynamics (CFD) analysis which offers a numerical solution to equations that define the flow variables. These equations are discretised and solved for small volumes called cells, which are distributed throughout the flow field and arranged as a mesh. Since the number of cells required for engine flow analysis is generally very large, of the order of a few hundred thousand, the calculations require fast computers with a lot of memory. The development of CFD as an analysis tool has therefore gone hand in hand with improvements in computer technology.

Computers are increasingly being used in industry and many companies now have the capability to perform in-house CFD calculations for practical applications. Industry's use of CFD is still restricted, however, sometimes by the computing resources required but more usually by the time required to define the geometry of the flow field and generate a computational mesh of that geometry. This bottle-neck in analysis means that whilst CFD may be involved in fault detection for a given design of engine, currently it may not be fast enough for use within the engine design process itself.

Section 1.2 describes the operation of an IC engine and how the flow field influences its

performance. As is also described in Section 1.2, the geometry of the engine affects the flow field and so the design of an engine's geometry is critical to its operation. The engine design process is described in Section 1.3, outlining how experiments could be integrated into engine design if they could be performed quickly and easily enough. Sections 1.4 and 1.5 describe physical and CFD experiments of engine flows respectively. In these Sections, the problems associated with the experiments are discussed. From this it becomes clear that CFD has the potential to play a significant role in engine design.

The work in this thesis explores methods for speeding up the preparation times for CFD analysis of engine flows and seeks to determine whether CFD can be used routinely in engine design by Jaguar Cars Ltd. Thus Section 1.6 describes the scope of this work and presents an outline of the thesis.

1.2 The problem considered

Section 1.2.1 outlines the workings of an IC engine and why details of the flow field are useful in engine design. In this work, one particular engine geometry is considered. In order to be relevant to industry, the engine geometry studied is taken from a real engine during its development and this geometry is described in Section 1.2.2.

1.2.1 IC engine operation

The four-stroke spark ignition (SI) IC engine operates on a cycle of induction, compression, combustion and exhaust. For a simple engine with two valves per cylinder, the cycle begins with the piston at top dead centre (TDC) and both the intake valve and the exhaust valve closed. During the induction stroke, the intake valve opens and the piston descends to bottom dead centre (BDC), drawing a mixture of air and fuel into the combustion chamber or cylinder. During the compression stroke, both valves are closed and the piston rises back to TDC compressing the air-fuel mixture. Ignition then occurs and the pressure rise associated with combustion forces the piston down the cylinder to BDC, so producing a torque on the

crankshaft and hence power. Lastly the exhaust valve opens as the piston returns to TDC and pushes the products of combustion out through the exhaust port.

Desirable attributes for an IC engine are good power output, fuel efficiency and complete combustion to minimise emissions which include oxides of carbon and nitrogen, and soot, as described by Arcoumanis and Whitelaw [2]. Good power output depends upon fast and even combustion of the compressed air-fuel mixture. Fast combustion makes the piston descend quickly giving good power output, whilst even combustion ensures that the piston descends smoothly. The latter minimises energy losses due to friction, extends the lifetime of the engine by reducing wear and also reduces stresses on the crankshaft. Complete combustion gives good fuel efficiency, which is an important factor not just for consumer costs but also for conserving fossil fuels. Complete combustion also uses most of the oxygen in the in-cylinder air. This prevents the oxygen combining with carbon and nitrogen to produce noxious oxides, notably carbon monoxide and nitrous oxides, NO_x, which would otherwise be emitted from the exhaust and pollute the atmosphere. It can therefore be seen that desirable IC engine operation depends upon the combustion process.

Combustion is the burning of fuel in the presence of air and to maximise this process the flame front must advance quickly and evenly (that is, at a steady rate) through the combustion chamber [1]. Flame development is described by Arcoumanis and Whitelaw [2], who state that the turbulent flame speed is related to the precombustion turbulence intensity. A turbulent flow field may therefore enhance combustion but excessive turbulence can hinder the exhaust process [1] and in some cases even blow out the spark and so prevent combustion. Turbulence levels therefore play a significant part in the operation of SI engines.

It has been well established [1,2] that the engine geometry significantly affects the in-cylinder flow field and turbulence levels [3]. Indeed, many experiments have demonstrated the engine geometry's affect on the flow field very well, for example, Lancaster [3] showed that in-cylinder turbulence levels are primarily due to the jet flow through the intake port, which in turn is influenced by the valve geometry. Similarly, Kastner et al [4] demonstrated the

valve's influence on the induction process. Vafidis [5] concluded that the in-cylinder swirl velocity distribution during the induction stroke is determined by the induction swirl levels and the geometric details including the intake geometry and the piston shape.

Engineers therefore need as much information as possible about the flow through a given engine geometry in order to design for good combustion in the engine.

1.2.2 Engine geometry used

The engine studied in this work is a four-valve reciprocating piston SI IC engine with a pent-roof to the combustion chamber. One large intake port splits into two separate intake ports, each with its own intake valve. Figure 1.1 shows the surfaces of these ports and the pent-roof.

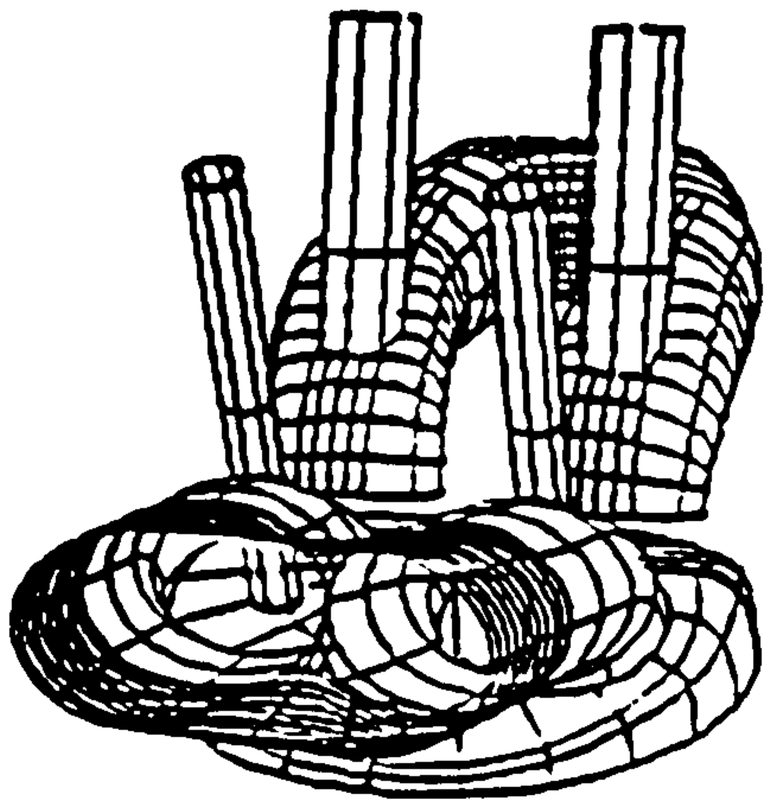


Figure 1.1 Surfaces of ports and pent-roof

The engine geometry was supplied by Jaguar Cars as computer-aided drawing (CAD) data and engineering drawings. The CAD data included surface details of the port and the cylinder roof geometry as would be machined from a block of the appropriate metal. Thus it described the shape of the cylinder roof, the inlet and exhaust ports, and holes that would later contain the valve guides and stems. To complete the port and cylinder geometry, the valves, valve guides and valve seats and the cylinder itself all had to be added to this data. Detailed engineering drawings of the valves and their guides and seats were therefore also supplied.

It was noted that a plane of symmetry existed through the cylinder axis. An assumption was made that the symmetric geometry would produce a symmetric flow field about the plane of symmetry. This meant that a computational mesh that describes only half of the port and cylinder geometry could be used in this work. This mesh would have significantly fewer cells than a mesh of the entire geometry. Since typical meshes of port and cylinder geometries require over one hundred hours spent performing calculations in the central processing unit (cpu) on a CONVEX mainframe computer, as used by Jaguar, any means of reducing the number of cells in the mesh will significantly reduce the time and resources used to obtain the CFD solution.

Also, this study only considers air flow during induction and compression, that is, cold flows simulating engine conditions before combustion. As such, the details of the exhaust geometry are not considered in this work. Thus the actual geometry modelled consists of one intake port and the roof and cylinder which are cut in two along the symmetry plane.

1.3 Engine design process

A simplified model of the design process from specification to manufacture is illustrated in Figure 1.2 below, taken from Rooney and Steadman [6].

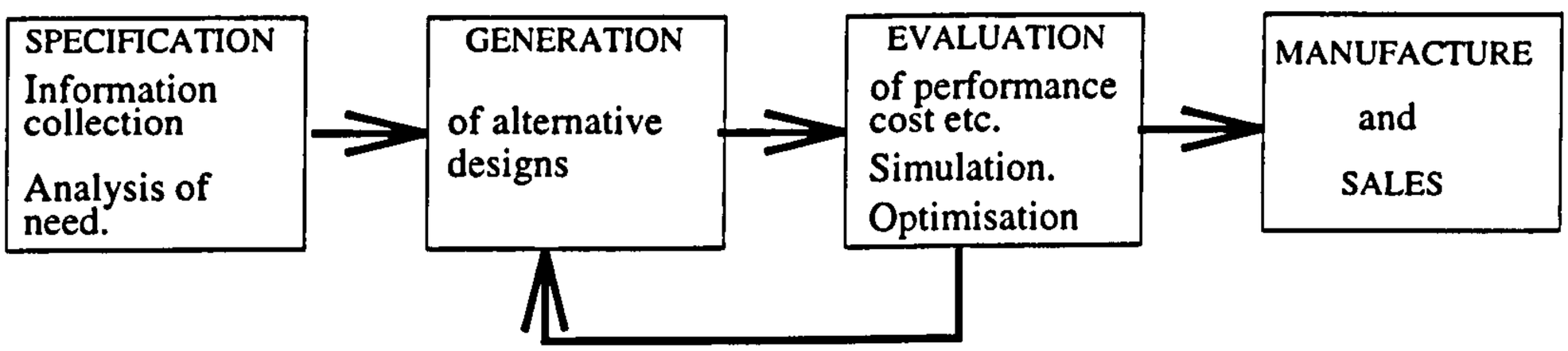


Figure 1.2 Model of the design process

As illustrated in Figure 1.2, once the design has been specified a number of alternative designs are generated. These designs are then evaluated against a set of parameters which include performance and cost. Once a suitable design is selected, the design is then

manufactured and the product is sold.

The generation of alternatives and their evaluation is shown as a loop in Figure 1.2, and this cycle may be performed a number of times until a suitable design has been found. The traditional design procedure for engines as described by Heywood [1] was to build the alternative designs and then test their performance. This approach could prove expensive and time-consuming if a number of prototypes were to be evaluated. Costs can be reduced, however, if experiments can indicate the performance of the design before prototypes are built.

For engines, performance is significantly influenced by the in-cylinder flow field which is in turn affected by the port and cylinder geometry, as mentioned in Section 1.2.1. The aim of some experiments in engine design is therefore to determine the in-cylinder flow field in terms of its flow structure. This will include axial swirl; the motion of the air and fuel mixture around the cylinder axis, barrel swirl or'tumble'; the swirl normal to the cylinder axis, and any flow separation that may occur, for example near the valve. Other useful data includes measures of pressure, fluid density and turbulence levels at TDC prior to ignition.

Once the flow structure has been determined, the results of more fundamental research into how the flow structure at TDC influences combustion is considered and the performance of that engine geometry can then be predicted. This prediction is an estimate, however, and the engine performance can only be fully determined by doing a full evaluation of the prototype.

The best experiments that can be performed are those which provide details of the in-cylinder flow field throughout the engine cycle, including the combustion process. Such experiments are difficult to perform, as will be outlined in the next Section. Even so, experiments into smaller parts of the cycle, such as examining the flow past the valve during induction, will at least yield a way of comparing different component designs before any prototypes are built. In this way, these experiments can still produce results which are valuable to engine designers.

The usefulness of experiments to designers depends upon how quickly they may be performed. Indeed, a set of experiments comparing different designs will be less valuable to the designers if they take longer to carry out than actually building and evaluating prototypes for each design. These experiments can be physical or numerical. Both physical and numerical experiments are discussed in Sections 1.4 and 1.5 respectively. The best method to use in the design process would be the most flexible, fast, cheap and reliable procedure.

1.4 Physical experiments

Experimental work into IC engine flows has been extensive during this century. Due to the complexity of these engine flows, however, many experiments have been performed using simplified geometries. In this way, researchers aim to isolate the geometric influences on individual flow features. Some such studies include studies of intake port flows by Brandstatter et al [7], flows past intake valves by Namazian et al [8] and Luo and Daneshyar [9] and in-cylinder flows by Bopp et al [10]. Examples of more complex flow studies include steady flow analysis of port and cylinder flows by Kastner et al [4] and by Cheung et al [11] along with studies of transient flows with moving pistons and valves by Vafidis and Whitelaw [12], Arcoumanis et al [13], El Tahry et al [14] and by Khalighi [15]. This is not an exhaustive list of all the experimental research published in this field, but it does highlight the broad range of engine flow experiments.

Levels of turbulence in engines prior to combustion have also been investigated, for example by Wakisaka et al [16] and Murakami et al [17]. Separate studies into combustion itself have been carried out using simple experiments, such as those by Armstrong and Bray [18], in which the fuel and air mixture is burnt as a jet in open air to explore the processes occurring at the flame front. More comprehensive studies have included both engine flow and combustion such as Alkidas et al [19] and Reuss et al [20], whilst others have studied the flow in a fired engine through the full engine cycle, Bopp et al [10] and Lorenz et al [21].

The closer to real engine geometries and operating conditions that the experiments come,

however, the more difficult the measurements are to obtain since engines are hostile environments particularly for delicate equipment, as described by Witze [22]. The work presented in this thesis considers cold flows and so a description of the experimental techniques used to study cold flows only is given in the following subsections.

1.4.1 Flow visualisation

Flow visualisation can be achieved with liquid or gas analogues of the engine. Coloured dyes are injected into the fluid which produces a series of streaklines that trace the course of the injected particles as they move through the cylinder. Particles seeded throughout the fluid can also show the fluid motion when used in conjunction with various forms of photography and video to capture their course, such as that described by Ekchian and Hoult [23].

A particularly successful form of flow visualisation is particle imaging velocimetry (PIV), for example Reuss [20], in which a laser emitting a thin sheet of light is shone into a transparent cylinder filled with a liquid with seeded particles. Two cameras at right angles to one another take high speed photographs. Computer software then interprets the results to determine both the flow structure at that cross-section of the flow field and the velocity of the seeded particles captured by the cameras. Indeed, this can also be done three-dimensionally, as has been described by Trigui et al [24].

In-cylinder flow can also be visualised by using Schlieren photography as used by Endres et al [25] and Alkidas at el [19]. The Schlieren photographs are short time-exposure photographs (or high speed movies) taken through quartz walls in the cylinder. Light is shone through the cylinder in parallel beams, but changes in the refractive index of the fluid deflects the light. The beams are brought to a focus on a knife-edge which stops the main beam. Any deflected light will pass this stop and the differing amounts of light passing the stop will produce local changes to the image captured on film. The refractive index of a compressible fluid changes with the fluid's density, and so the deflection of the light may be caused either by changes in temperature or in velocity. Therefore the Schlieren photographs can show

regions which have different speeds within the fluid, which makes it a powerful technique for visualising the structure of the flow field.

1.4.2 Velocity measurement

Whilst most flow visualisation techniques only show the flow structure, the exception being PIV, other experimental methods may be used for measuring flow velocity. The most popular methods used for this are laser doppler velocimetry (LDV) which has been used by Brandstatter et al [7], Cheung et al [11], Vafidis and Whitelaw [12] and Arcoumanis et al [13] amongst others, and hot-wire anemometry (HWA) as used by Lancaster [3], Luo and Daneshyar [9], El Tahry et al [14] and Wakisaka et al [16] amongst many others.

The LDV method measures the scattering of laser light as it passes through specially built engine cylinders which have optical access via quartz windows. As a seeded particle passes through the focus of a laser beam it interferes with the wave pattern and reflects the light at different angles. The deflection of the light from the laser's path is proportional to the velocity of the particle passing through the beam. Hence statistical analysis of the scattered light reveals both the mean velocity and the deviation from the mean velocity, that is, the turbulence at this point in the flow.

The HWA technique involves placing a thin wire at various positions in the flow domain. An electrical current is passed through the wire and the resistance of the wire generates heat. The air passing over the wire cools it, and so the higher the velocity, the higher the cooling rate of the wire. Since the resistance of the wire changes in a known way with its temperature, the speed of the air flowing over the wire can therefore be determined by finding the electrical resistance of the wire or by finding the power required to keep the temperature of the wire constant. By using two or three wires perpendicular to one another the components of the air flow velocity in three directions can be measured, which gives a measure of the air speed and direction.

1.4.3 Swirl measurements

The flow about the cylinder axis is called swirl and it can be quantified using a light paddle wheel or an impulse swirl meter in steady flow experiments. These measure the torque exerted by the fluid on the wheel and derive a swirl coefficient which compares the flow's angular momentum with its axial momentum as described by Arcoumanis and Whitelaw [2].

Tumble, that is, the flow about an axis normal to the cylinder axis can also be quantified using swirl meters. In this case a tube is attached to the side of the cylinder, normal to the cylinder axis, and a swirl meter is placed inside. In theory, if angular momentum is conserved then the swirl meter will register a value proportional to the in-cylinder tumble. In practice, however, this is not the case since the angular momentum is not conserved. Some studies have shown a correlation between the in-cylinder momentum and the tumble measured by the swirl meter in this way [26].

Such measurements of swirl and tumble would only be accurate if the axis of the meter coincided with the axis of the rotating flow. The in-cylinder flow is highly complex, however, having different directions and scales of vortices throughout the cylinder. Thus swirl meters are very crude tools which will usually only indicate the general momentum in the flow.

1.4.4 Problems of physical experiments

Sections 1.4.1 to 1.4.3 have presented the commonest measuring techniques used in the analysis of engine flows. None of these techniques allows pressure, velocity and turbulence to be measured at the same time, however. Generally, different experimental rigs are required for the different techniques. Data collected from these experiments are also limited by the number of measurements made. Indeed, where experiments are limited by time constraints, the number of measurements made must be limited and the location of the measurements taken becomes very significant.

Another problem in physical experiments is the engine geometry itself. For example, regions near the valves can be inaccessible for LDV measurements. Hence simplifications of the geometry are often made which will give different results from the real engine, although they can be very useful in a general sense. Furthermore, the conditions of engine operation make things more difficult for the experimenter. Engine flows are transient with high velocities and accelerations of the flow. In a realistic operation, the engine is also very hot, subject to powerful explosions and produces exhaust products which make visualisation very difficult. Under these conditions, delicate instruments can be damaged or used only once.

For these reasons, many experimental engine studies are made of the flow field without combustion to determine turbulence levels and flow features, which rely on separate studies of combustion to determine the interaction between turbulent features and combustion.

1.5 CFD experiments

CFD experiments into the fluid flow in complex geometries has become viable in terms of time and cost within the last ten years or so. There have been rapid and continuing developments in computer technology, as described by Girdinio et al [27], in CFD modelling such as described by Watkins et al [28] and in CFD codes during this period. This means that the use of CFD as a design tool is fast becoming a reality as described for automobile aerodynamics by Kobayashi and Kitoh [29] and CFD has an increasing role to play in all aspects of automotive design and development, as discussed by Gosman [30]. CFD methodology is outlined in Section 1.5.1 and then the application of CFD to engine flows is described in Section 1.5.2. Finally Section 1.5.3 details some of the problems found in CFD analysis.

1.5.1 CFD methodology

The use of CFD for fluid flow analysis involves a number of stages, as described in detail by Shaw [31]. Firstly the flow problem must be considered in context, to determine the geometric details of the flow domain and what type of flow is to be analysed. This is

essentially a planning stage, and consideration must be made of any existing description of the flow domain, such as CAD data or engineering drawings.

Next the flow domain must be divided up into a computational mesh consisting of elements or cells. The outer faces of the cells in the mesh are called boundaries. Flow conditions at the boundaries can be defined easily by grouping boundaries together into regions and then applying the flow conditions to each region. Boundary flow conditions relevant to engine studies include flow inlets, outlets and walls. Numerical values for the flow parameters such as initial and boundary conditions must then be defined for the mesh and then a solution calculated. Finally the results must be analysed to provide information that is useful to the user. In many analyses, this whole process has to be iterative, with changes made to the model to improve the results, as is shown in Figure 1.3, taken from Shaw [31].

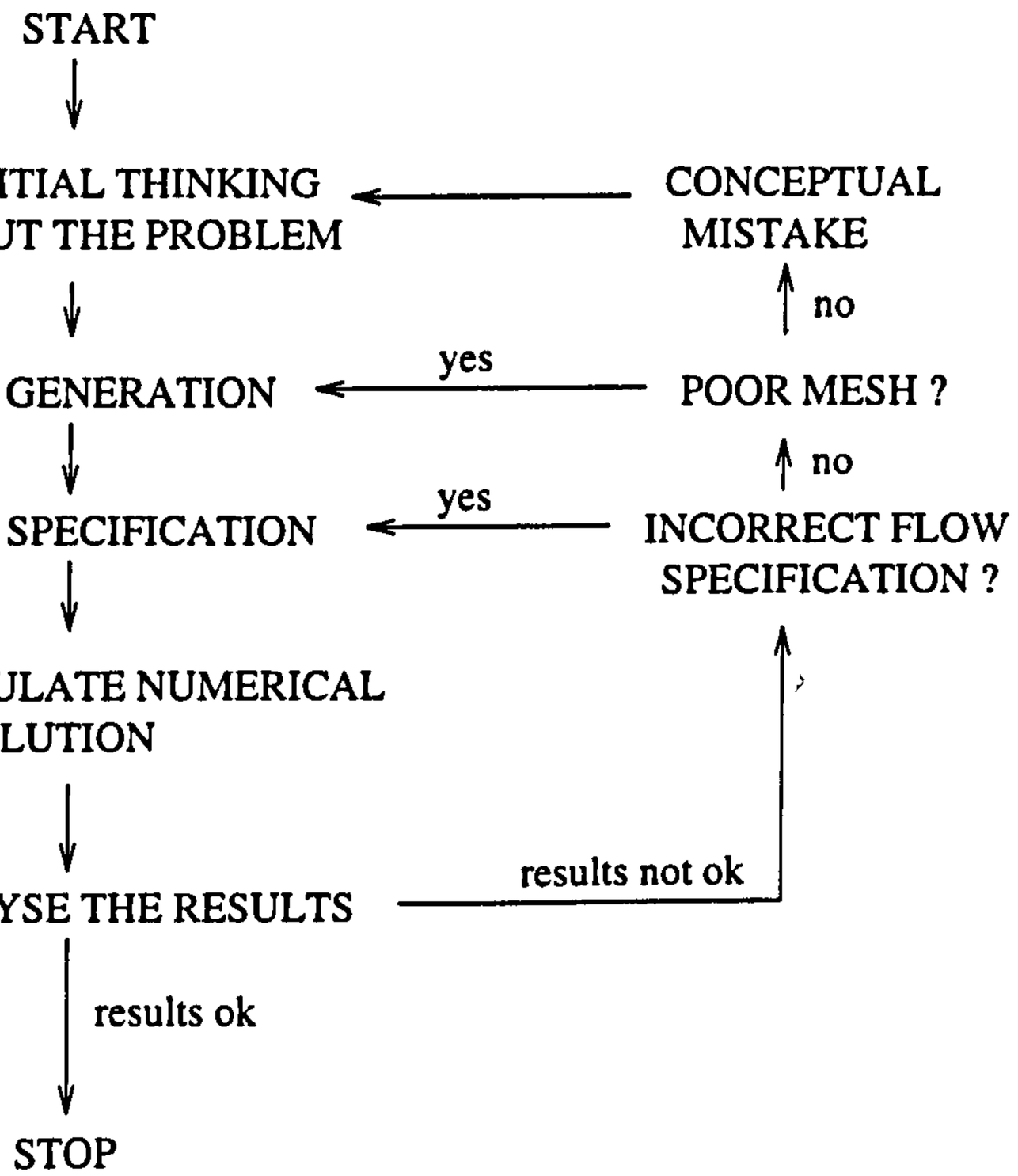


Figure 1.3 The CFD method

Computer software packages are used for some of the stages in CFD analysis. The definition of the flow domain and its division into a grid or mesh of cells requires a 'pre-processor'; a software tool with which the flow domain is defined. The flow parameters are also entered into the pre-processor and this data is all passed to a CFD 'solver' which calculates a solution for the flow problem. Finally the results are passed to a 'post-processor' which uses various graphics techniques to display the results. Graphics capabilities are required both in pre-processing to view the computational mesh and in post-processing to view the results. As a result, pre- and post-processors are often combined into one package in commercial software.

Meshes can be unstructured or structured, although the definitions for these are not yet standardised in the literature. A fully structured mesh has quadrilateral cells in 2D or hexahedral cells in 3D, and the overall mesh will be topologically a quadrilateral in 2D or a hexahedron in 3D. In particular, structured meshes have an equal number and distribution of vertices on opposite faces of the mesh and a regular vertex labelling system for the coordinate directions of that mesh. Structured meshes can be deformed to fit a geometry and may also be referred to as body-fitted meshes. Unstructured meshes may contain any cell type, but can also describe meshes containing only hexahedral cells that are not arranged in the regular fashion of body-fitted structured meshes. A special case of this is the 'block-structured' mesh in which a set of structured meshes are assembled to form a single mesh that is not itself structured.

CFD solvers calculate the flow variables of a fluid flowing through a given domain. These flow variables include viscosity, density, pressure and fluid velocity and they are solved for both laminar and turbulent flows by numerically modelling the continuity, momentum (Navier-Stokes (NS)) and energy equations, as described by Shaw [31]. The continuity equation is a partial differential equation (PDE) describing the conservation of mass in terms of the fluid velocity and density. The NS equations are also PDEs which describe the conservation of momentum of a fluid in terms of velocity, density and viscosity. Although the continuity and NS equations can be used to analytically calculate the flow variables for laminar flows in very simple geometries, this is not the case for turbulent flows. When turbulence

occurs, direct computer simulation can be used, but in complex geometries the computational effort required is currently prohibitive. Therefore simplified numerical models of turbulence are included in commercial CFD codes.

The development of mathematical models of turbulence continues, but the most common turbulence model in commercial use is the 'kinetic energy - dissipation' or 'k-ε' model described by Launder and Spalding [32]. In the k-ε model a turbulent viscosity is assumed, which is calculated from values of the turbulent kinetic energy, k, and its dissipation rate, ε. PDEs are formulated to describe the variation of these variables, but they are simplified by replacing some of the terms in the equations with constants derived from experimental data. Whilst this gives a fair approximation of the turbulence in a flow, it is not entirely accurate for every flow case, especially if the flow geometry or the streamline is curved, and so some error in the solution obtained from the CFD codes must be taken into account.

The NS equations can only be solved analytically for the very simplest of fluid flow problems, and so for complex flow problems they must be approximated numerically by discretisation. Many commercial CFD codes use the finite volume method to discretise the PDEs governing the fluid flow, as described by Patankar [33]. These PDEs contain first- and second-derivatives of the flow variables with respect to time and spatial coordinates. The time-dependent terms are discretised by using forward differences. The space-dependent terms are then discretised by considering each volume or cell of the mesh in the flow domain. The second-derivatives of the flow variables are stored for the centroid of a cell and can be expressed as the differences between the first-derivatives of the variables which are stored for the faces of that cell. The first-derivatives of the flow variables at the faces of the cell, can in turn be expressed as the difference between the values of the flow variables themselves, which are stored for the centroids of the neighbouring cells. Thus for one cell, the PDEs that describe the flow variables are discretised into a set of difference relations of the flow variables. These discretised equations are found for each cell in the mesh, and assembled into a set of simultaneous equations which must be solved.

The flow equations contain non-linear terms, and so cannot be solved directly in their discretised form. Instead the solution is obtained using iterative procedures.

1.5.2 CFD analysis of engine flows

IC engines have been a major challenge for CFD due to the complex geometries of the flow domain and the in-cylinder processes, as discussed by Gosman [34]. Engine studies using CFD can include compressible, turbulent flow with dispersion of fuel particles within the air, variable geometry with the moving piston and valves, chemical reaction as the fuel is burnt and heat transfer effects as the engine becomes hot. Echoing the trend of experimental work, many CFD studies have been carried out for cold flows, that is, without heat transfer or chemical reactions.

Similar again to the experimental trends, CFD analysis for internal combustion engines have frequently separated flow studies into the strokes of the engine cycle, with simple in-cylinder flow studies using experimental data for valve curtain boundary conditions such as the work presented by Yamada et al [35] and by Gosman and Johns [36]. The cylinders modelled have tended to be axisymmetric and so the computational mesh for such a geometry is much easier to create than for a full 3D engine geometry. Also, body-fitted structured meshes could be used with finite volume codes, as will be discussed more fully in chapter 5, keeping the demands upon computer memory and run times low.

A further simplification that can be made in the study of axisymmetric geometries, is to investigate a 2D 'slice' of the cylinder such as Bracco and Grasso [37] and Ramos and Sirigano [38]. Using simple grid structures, studies of moving pistons can also be performed for in-cylinder flows using valve-curtain data obtained from experiments as boundary conditions for example Gosman et al [39], Wakisaka et al [40] and Kojima and Takata [41].

As computers have improved in recent years [27], it has become viable to use larger and unstructured meshes and consequently CFD engine flow studies have been extended to

include the port-valve-cylinder flow for example Aita et al [42] with transient cases using moving meshes for example Naitoh et al [43], Errera [44] and Taghavi and Dupont [45]. As with any predictive techniques, CFD results have had to be verified by comparison to experimental results, and many studies have included parallel CFD and physical experiments which test the capabilities of CFD models and codes for example Le Coz [46], Amsden et al [47] and Gosman et al [48].

1.5.3 Problems of CFD experiments

The main problem in performing CFD experiments today is the time and skills required to generate a computational mesh of a complex geometry. The development of fast automatic mesh generators continues but as will be discussed in chapter 5, a general, comprehensive commercial code has yet to appear on the market.

A further problem for the acceptance of CFD in design is the errors in the CFD analysis itself, which can occur for a number of reasons. Firstly the mesh must be fine enough to allow all significant flow features to be calculated. Since the numerical equations are solved for on a cell by cell basis, it follows that flow features which are smaller than the cell size cannot be detected. Flow velocities and gradients may also be affected by a large grid size and these gross errors may again be resolved by using a finer grid. This, then, is a question of proper problem formulation rather than an inherent CFD problem.

Errors in the CFD calculations also occur and these include rounding errors, numerical approximations and mathematical modelling. As with any iterative calculations, rounding errors can occur and be propagated through the solution or be increased by successive rounding errors. Many codes have the capability to work in 64 bit 'double precision' mode on the computer, and this will help to minimise the affects of rounding errors.

On top of this, the discretised flow equations are approximated numerically so that they can be solved iteratively, which introduces the error of approximation. Finally, in discretising the

flow equations, some terms describing the average of fluctuating variables are left out, and these are the terms which describe the turbulence in the flow. Thus mathematical models of turbulence have been created to reintroduce the turbulent component of the flow into the calculations. These are empirical formulations and as such only truly apply to the flow conditions under which they were formulated. Any other flow regime will not necessarily be properly described by these turbulence models and so further errors are introduced into the results.

These errors in CFD mean that validation of the codes for a given flow field is important. For this reason, CFD will not completely replace physical experiments in the future rather CFD has a significant and complementary part to play in the design process.

1.6 Present contribution

The overall aim of this work was to determine whether CFD could be performed quickly and accurately enough so that is could be used routinely in engine design. There are two ways in which CFD may be used in the design process. First, CFD could be used as an integral part of generating a number of alternative designs. Here new concepts would be tested early on to determine whether they are viable designs. A number of different geometries can then be compared and the most promising few selected for evaluation using prototypes. In this role the CFD analysis should have an overnight turnaround time in order to be most useful to the designers.

The second way in which CFD could be used in design is as a fault finding procedure. In this case the designers generate a number of designs and decide which ones are most promising based only upon their knowledge and experience. CFD may then be used to test the design either before the prototypes are built or at the same time, and effectively becomes another tool for evaluating the prototype. Thus CFD may detect flow related faults whose cause cannot necessarily be found using engine testing. The turnaround time for the analysis is not crucial in this case, although it should be fast enough for the generate-evaluate loop to be run a number of times.

The three months typically required to perform a steady flow CFD analysis of an engine at the beginning of this work [49] was unsuitable for either ways of using CFD in engine design. The best situation would be for the analysis to run overnight so that CFD could be used in all aspects of engine design.

A number of aspects of CFD analysis were therefore examined as follows;

- creating the geometry
- generating the mesh
- benchmarking the results
- integrating CFD into current engine design practices at Jaguar Cars Ltd.

The intention of this work was to establish a suitable method for performing CFD as an integral part of the engine design process. Ideally, the CFD analysis could be performed automatically or by a design engineer rather than a CFD specialist, although this would leave questions of judging the quality of the results unresolved. Even using specialist CFD analysts, the CFD method recommended must be the quickest and easiest to use given the hardware and software available within Jaguar. Additionally, if the existing hardware and software were found to be unsuitable, recommendations for alternatives were to be made.

During the course of this work, the main problem with performing CFD analysis was found to be the long time required to generate a computational mesh of the complicated engine geometry. In an attempt to address this problem, a novel method for mesh generation was developed by the author. In this method, the user only generates meshes for simple geometries which is generally a quick and easy operation. Software designed by the author then automatically builds up the complicated geometry from these simpler meshes both by linking meshes and by cutting one mesh out from another.

1.6.1 Objectives

The original objectives for this work were as follows;

1. Design a procedure for creating a complete CAD model of the engine geometry based upon typical engine data from a number of sources within Jaguar.

2. Design a procedure for creating a computational mesh from this geometry which minimises the time and skills required to create the mesh, but which also considers the quality of the resultant mesh.

3. Prove this procedure using a current development engine geometry and perform steady state CFD analysis.

4. Validate these results against experimental data to assess the quality of the results.

5. Establish a procedure for performing transient analysis of the port and cylinder geometry with a moving piston and valves.

In attempting to meet the second objective, the author designed a mesh construction program. The objective of this program was also to reduce mesh generation times and make mesh generation easy.

1.6.2 Outline of Thesis

The work presented in this thesis is organised as follows. Chapter 2 describes the investigation of a method for taking CAD data and creating a computer model of the engine geometry. Chapter 3 outlines mesh generation, exploring various possible routes to obtain a mesh and the creation of various types of meshes. The CFD analysis using these meshes is described in chapter 4, including the benchmarking of the results with respect to experimental measurements performed at Imperial College, London. The implications for using CFD within Jaguar's current engine design process are also discussed.

An overview of various automatic mesh generation techniques follows in chapter 5. The

author's novel method for both linking two meshes together and cutting one mesh from another is presented in chapter 6. The author also wrote a computer program to perform these mesh manipulations with as little user intervention as possible, and test cases for this program are given in chapter 7. Chapter 8 then presents the conclusions of the thesis.

2. CREATING A MODEL OF THE ENGINE

2.1 Introduction

An important part of CFD analysis is the definition of the geometry to be meshed. For simple geometries, the mesh can be generated easily using, for example, the basic meshing tools in the CFD pre-processor. The mesh generation tools in commercial CFD packages have been generally limited in the past, although improving mesh generation facilities is now a major part of many commercial package development. Even so, for more complex geometries other tools must still be used. In this situation, a computer model of the flow domain should be created, from which a mesh can then be created. This chapter describes the creation of the computer model in preparation for mesh generation.

The hardware and software used in this work is outlined in Section 2.2 and the creation of CAD data is then described in Section 2.3. In order to create a complete computer model the CAD data must be transferred to a different software package and the methods explored for doing this are presented in Section 2.4. The ways in which a computer model can be created are outlined in Section 2.5. Section 2.6 describes the methods for creating a computer model for each component in the engine geometry. The assembly of these components into a single computer model is then presented in Section 2.7. Section 2.8 describes creating the engine geometry from CAD curves. A summary of the models created using these techniques is then presented in Section 2.9. The implications for Jaguar are discussed in Section 2.10 and a summary of the chapter is given in Section 2.11.

2.2 Computing resources

The hardware and the software used in this work were selected in order to tie in as closely as possible with the computing resources used by Jaguar in their Advanced Engineering Centre. The hardware and software were installed at Warwick University and were judged initially to be adequate for the task.

2.2.1 Software used

Three commercial software packages were used for in work;

CV CADDS, a CAD package from Computervision

I-DEAS, a general engineering analysis package from SDRC

STAR-CD, a CFD code from Computational Dynamics Ltd

CV CADDS was used by Jaguar to create a CAD model of the engine geometry. Using a CAD package, a three-dimensional computer model of a given geometry can be built up from a set of points, curves and surfaces. This geometry can be viewed from any direction and so one CAD 'drawing' can represent a complete assembly. However the traditional engineering drawings can still be produced directly from the CAD data if required. The strengths of CAD lie in this replacement of a whole set of drawings, in the rapid manner that the information can be disseminated throughout a company, and in the ability to alter parts of the geometry without having to redraw the whole assembly. Furthermore, CAD data can be linked to numerically controlled (NC) cutting tools and other applications of computer-aided manufacture (CAM). CAD/CAM can therefore assist companies in shortening development times and hence cutting costs. CAD models can also form the basis for numerical analysis such as finite cell structural stress analysis or indeed CFD analysis as is proposed in this work. CV CADDS is one of the many commercial CAD packages available to industry and it was already being used at Jaguar. Thus the integration of CFD into the design process had to take account of CV CADDS.

I-DEAS was used for its model preparation and advanced mesh generation capabilities. I-DEAS contains a number of separate modules which are groups of commands and operations related to one application of the code. These modules include the Solid Modelling (SM) and Finite Element and Analysis (FEM) modules. SM is an object driven geometry modeller, which means that a geometry defined by a set of points, curves and surfaces is stored as a single solid 'object'. The object is operated upon as if it were a single entity making SM a very powerful modelling tool by simplifying the creation and manipulation of the computer model

of a geometry. In creating solid objects, SM uses a set of 'primitives' which are elementary shapes such as block, cylinders, tubes, cones and spheres that can be created with just one command. Three-dimensional objects can also be created from 'profiles', which are 2D cross-sections of a geometry, either by creating a set of surfaces between a group of profiles, a process known as 'skinning' or by rotating the profile about an axis. In the latter case, the path followed by the profile points define curves from which surfaces are then created to generate a 3D axisymmetric solid object.

The I-DEAS FEM module also performs some geometry modelling, but does not operate on objects. Rather, FEM manipulates points, curves and surfaces individually, much more like a CAD drawing package. In contrast to general CAD packages, however, the geometric data in I-DEAS has a hierarchical structure that imposes certain constraints on the geometric description of points, curves and surfaces. For instance, the curves which bound a surface must form a closed loop with the end-point of one curve also being the start-point of the next curve in the loop. Similarly, adjacent surfaces must share the mutual curves that lie between them. To ensure these conditions are met, I-DEAS uses a concept of associativity. Thus a point will be defined not only by its coordinates, but will also be associated with the curves for which it is an end-point, and with the surfaces that these curves bound.

FEM is primarily a mesh generator, however, that has been designed for structural stress analysis. Curves outlining the geometry are used to define surfaces known as 'mesh areas' and these surfaces are used to define 3D blocks referred to as 'mesh volumes'. These mesh areas and mesh volumes are then filled with vertices and cells to form a mesh of the geometry.

I-DEAS has a relational database (PEARL) to transfer data between its own modules and other software packages. Universal files [50] contain data for the computer model in an I-DEAS defined format and are used to transfer data between different software packages.

During the course of this work, three versions of I-DEAS were used and each successive

version had new or improved functionality. Thus some problems encountered when using I-DEAS Level IV at the beginning of this work were solved by subsequent releases of the software; I-DEAS V and I-DEAS VI. This changed the course of the work, but did not prevent a full assessment of the methods available for geometry manipulation, as will be discussed further in this chapter.

The STAR-CD CFD analysis package has two modules; PROSTAR, a Pre/Post-processing program and STAR, the CFD solver. Like I-DEAS, PROSTAR pre-processing has modelling and mesh generation capabilities, but unlike those in I-DEAS, these are laborious to use for all but the simplest models. A mesh is generated in PROSTAR by first specifying each vertex in the mesh, listing its position in a 3D coordinate system and its unique label. Each cell is then defined by listing the vertices that lie at its corners in a specific order. PROSTAR has the capability to copy, reflect and rotate vertices and cells, which can speed up the mesh generation, but even for simple cases, each vertex and cell label and position must be determined before the mesh is generated. For complex geometries this becomes a difficult task and for typical engine geometries, where meshes of more than 200,000 cells may be required, meshing in PROSTAR becomes extremely time consuming. In order to compensate for these limited meshing capabilities, PROSTAR can import meshes generated from other commercial software packages such as I-DEAS, ANSYS, PATRAN and NASTRAN.

2.2.2 Hardware used

At Jaguar's Advanced Engineering Centre, engineers used Computervision CAD workstations for CAD drawings, a VAX mainframe computer which could be accessed using a number of remote terminals, and a CONVEX which could also be accessed remotely. Thus the network used by Jaguar links all the workstations, the VAX and the CONVEX together. Engineers at Jaguar used the FEM I-DEAS module for FE stress analysis, but SM was not used since it was considered to be too memory intensive to be run over the network and then its use would have slowed down the network access speed for all users.

In order to minimise any disruption to operations at Jaguar, the bulk of this work was performed at Warwick University. The computing facilities installed at Warwick University had to be capable of running I-DEAS and STAR-CD. I-DEAS required good graphics capability for geometry modelling and for mesh generation and also a large amount of processing power and computer memory for the solid modelling operations. In addition, the CFD models to be run in STAR-CD were expected to be large and so sufficient disk space would also be required to store these models.

In order to meet these requirements, a Sun 3/80 workstation was selected that had 8Mb Random Access Memory (RAM) and a 200 Mb hard disk. At the beginning of this work, I-DEAS Level IV was used, but during the course of the work Levels V and VI were released. I-DEAS Level V required another 8Mb RAM to be added to Sun 3/80 due to increased capabilities. I-DEAS Level VI could not be run on the Sun 3 series of computers and so had to be run using a Sun 4 Sparcstation. Such continued upgrading of computing hardware is not unusual in industry, and it makes purchasing computers a considerable problem for a company since an expensive computer may become obsolete within a few years.

2.3 CAD Data Creation

CAD drawings of parts of the engine geometry were created using CV CADDS software by engineers at Jaguar. The CAD models were created for CAD/CAM purposes and not specifically for analysis, indeed at that time the CAD draughtsmen were not aware of the type of information required by analysts.

Points and curves defining the geometry were created first, and surfaces were then defined from these bounding curves. The CAD data consisted of points, curves and surfaces which described the intake port and the roof of the cylinder as listed below;

surface data in CADDS:	ports, roof and valve guides
curve data in CADDS:	ports, roof, valve guide and valve seat
engineering drawings:	valve profile
dimensions only specified:	cylinder

The surface data from CADDS is shown in Figure 1.1 whilst the curve data is shown in Figure 2.1.

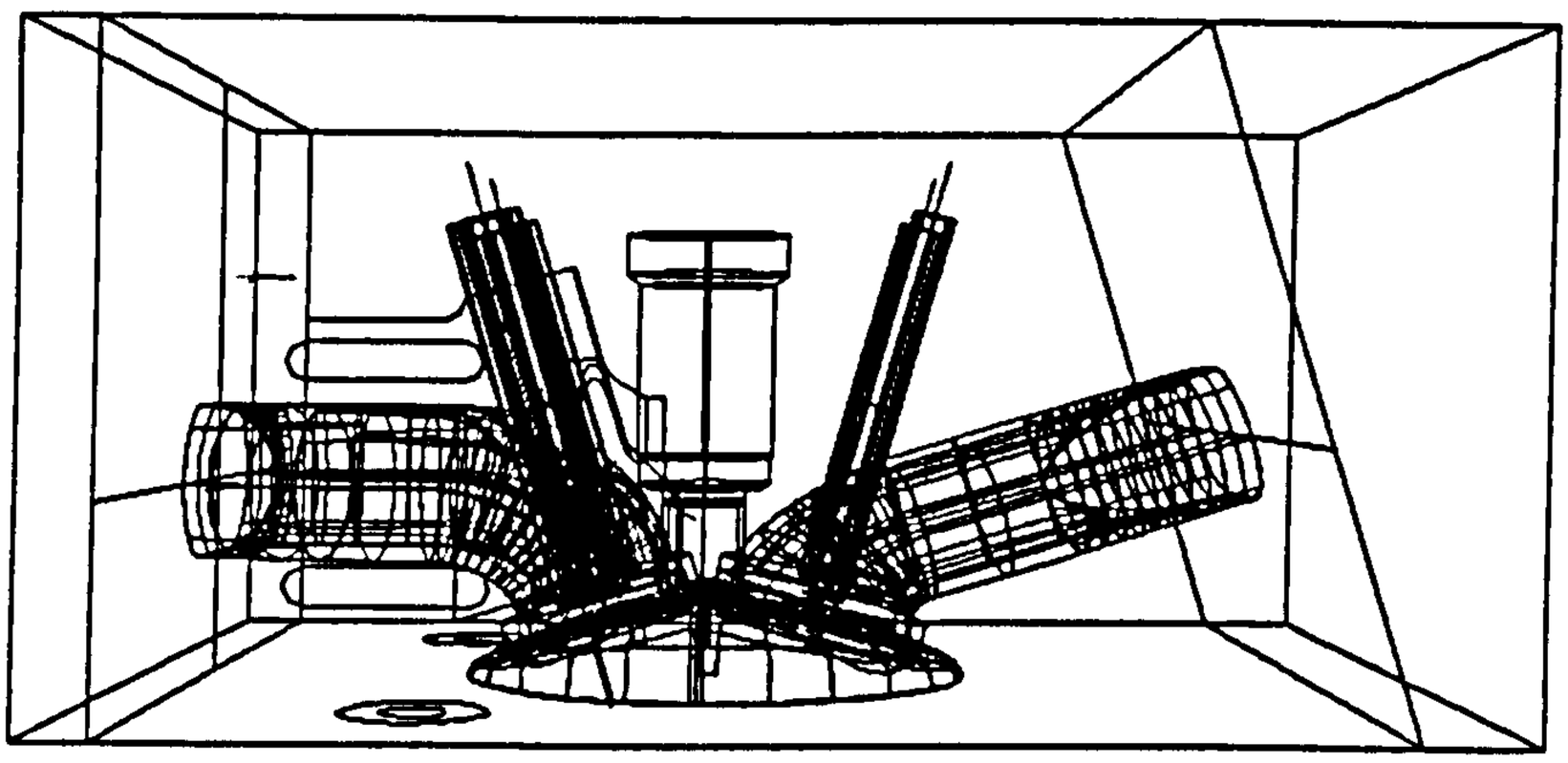


Figure 2.1 CAD curves created in CADDS

Such a scattered description of the engine geometry is typical of CAD usage in industry and is entirely sufficient for CAD/CAM purposes. The geometry required for NC machining is produced as surfaces, whilst other components are simply marked in position as line details. In order to create computational meshes for numerical analysis techniques such as CFD, however, a complete model of the entire geometry is required, defined as a set of 3D surfaces.

2.4 Transferring Data from CADDS to I-DEAS

Transferring the geometry data from CADDS to I-DEAS at Jaguar involved a transfer of data

between computers as well as between different software packages. This situation frequently occurs in industry where a mix of computer types can exist, even within a single company. The main problem with this is that operating systems and file formats may vary between makes of computers. In addition to this, commercial CAD packages often have different data handling and storage techniques and different mathematical descriptions of curves and surfaces from one another. To address this problem, an internationally recognised standard file format for transferring graphics data has been established called the Initial Graphics Exchange Specification (IGES). Ideally, graphics data written in IGES format can be transferred between different computers and between different software packages. In this work, the CAD data was written to an IGES file for transfer between software packages and stored on a 3/4" cartridge tape for transfer between computers.

I-DEAS provides different methods for reading an IGES file into its various modules and so the best method for Jaguar's purposes had to be identified. The aim of the data transfer was to have the CAD data in the FEM module where a mesh of the geometry would be created. The IGES file could not be read directly into FEM, but needed to be read through PEARL or SM as shown in Figure 2.2.

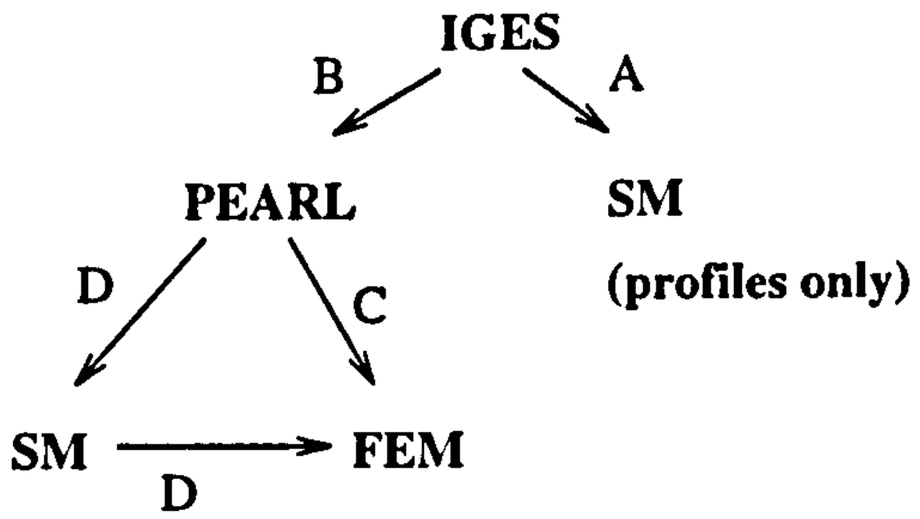


Figure 2.2 Routes for reading IGES files into I-DEAS

Only profiles could be read directly into SM from an IGES file. Since profiles were not included in the CAD description of the engine geometry, route A could not be used and so the IGES file had to be read into PEARL first of all, taking route B shown in Figure 2.2.

Once the CAD data was stored in PEARL, it could be read either directly into FEM for mesh generation, route C, or it could first be read into SM, modified, and then read into FEM, which is route D in Figure 2.2.

The CAD data needed some modification before a mesh could be generated from it, however, as is described in detail in the Section 2.5 below. This modification could be done in SM or FEM depending on which mesh generation methods were to be used and these are described later in Section 3. Therefore Sections 2.5 to 2.9 outline creating computer models both in SM and in FEM.

2.5 CADDS Data in I-DEAS

Ideally, the CADDS data could be used directly in I-DEAS to generate a mesh of the flow domain, but some essential differences between CV CADDS and I-DEAS geometry descriptions prevent this. Associativity as defined in I-DEAS does not exist in CADDS, and so the same geometry may be created in different ways in the software packages. For instance, a surface created in I-DEAS must have the endpoints of each of its bounding curves lying in the corners of that surface. There is no such constraint in CADDS, however, and Figure 2.3 illustrates this, where the surfaces labelled s1 to s8 were all created with boundaries lying on only part of a common curve labelled C20.

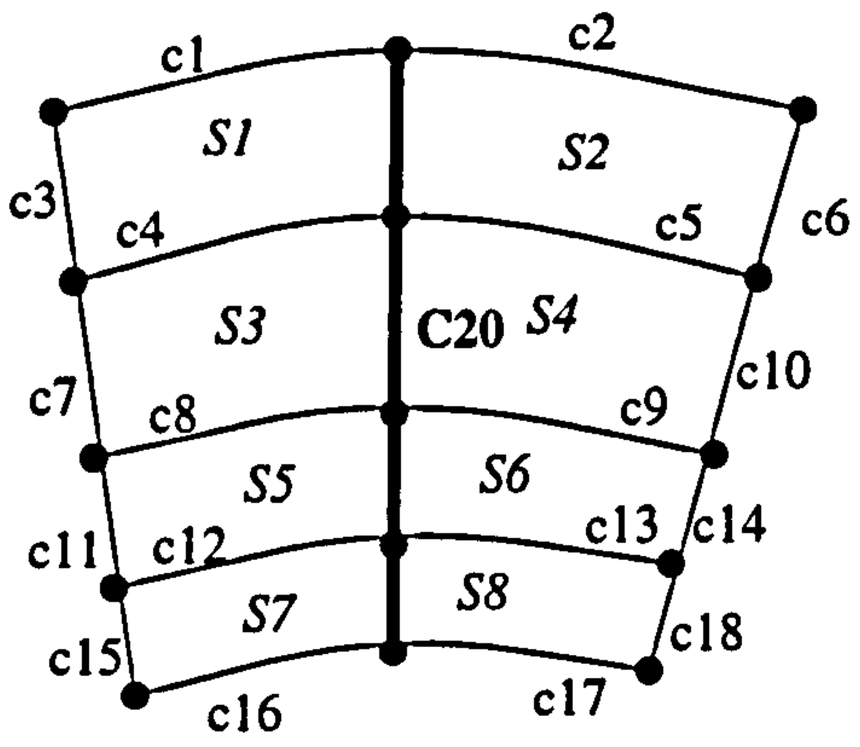


Figure 2.3 Eight surfaces created in CADDS using the same curve

The CADDS model of the engine geometry contained many such structures that were not acceptable to I-DEAS, and so simply adding I-DEAS associativity datasets manually would not solve this problem. Indeed, when the CADDS surfaces were read in to SM they did not have surface-curve associativity, and so could not be recognised in SM as defining a single object. Instead, each surface was treated as a separate solid object.

Another problem in matching the CADDS data to I-DEAS requirements was that the CADDS data had been created for use with NC machining tools. As such, the CADDS surfaces could contain slight gaps or overlaps between surfaces within the tolerances acceptable to the machine tools. Indeed, for areas of the geometry which would be cut by NC cutting machines, the surfaces could overlap greatly without adversely affecting the NC operation. The geometry for any analysis such as CFD, on the other hand, requires surfaces that match exactly. Furthermore, adjacent surfaces in I-DEAS are required to have boundary curves that match up, that is, only having one common curve between them. The CADDS data, however, had a number of sets of adjacent surfaces that did not match up in this way.

In view of these problems, a method for modifying the CADDS data or for creating a new computer model in I-DEAS had to be found. Three approaches to creating computer models were attempted, as illustrated in Figure 2.4 below.

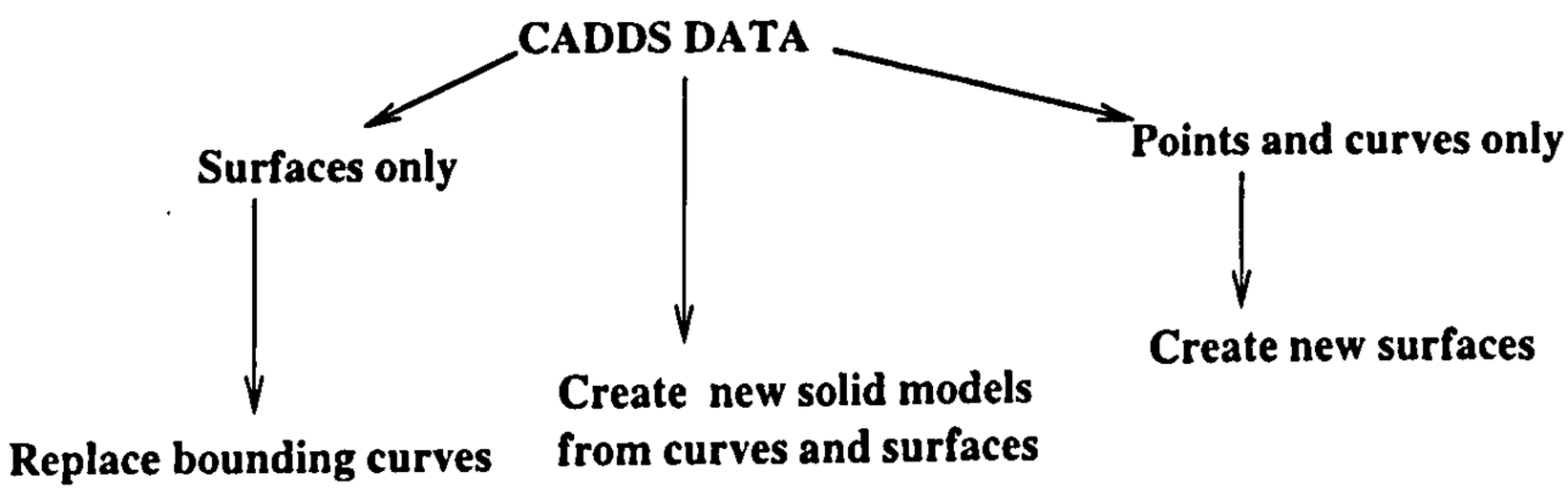


Figure 2.4 Using CADDS data in I-DEAS

From the initial CADDS data the surface data can be extracted and a new set of curves and points formed on their boundaries. Alternatively, the CADDS data can merely supply geometric dimensions and an entirely new solid model may be generated in I-DEAS. Thirdly, the points and curves in the CADDS data can be extracted and new surfaces can be created in I-DEAS.

In the first approach using only the CADDS surfaces, all the points and curves were deleted from a model file which contained the CADDS data, leaving only the surfaces. Bounding curves were then defined automatically in I-DEAS for every surface. This should have created points and curves on the boundaries of these surfaces with all geometric associativities included. Unfortunately, this operation did not create the desired set of bounding curves in I-DEAS Level IV; although the curves were associated with the surfaces, they rippled along the surface edges rather than following them accurately. This option was therefore rejected.

The second option was to use the points and curves of the CADDS data as geometric guides and templates for the creation of individual solid models of the components themselves, such as the port, roof and so on. The solid models of the components would be joined together in I-DEAS SM to create a single model of the geometry. The creation of component solid models is described in Section 2.6 and their assembly into a single solid model is described in Section 2.7.

A third option was to use the CADDS points and curves as a guide to creating a new set of surfaces in I-DEAS. This method is described in detail in Section 2.8.

2.6 Component Modelling

The engine geometry was divided into the following regions;

port and valve seat

pent-roof

valve and valve guide

cylinder

Solid model of these 'components' are described in the following subsections.

2.6.1 Creating a solid model of the port and valve seat

The methods explored to create a single solid model of the port are illustrated in Figure 2.5.

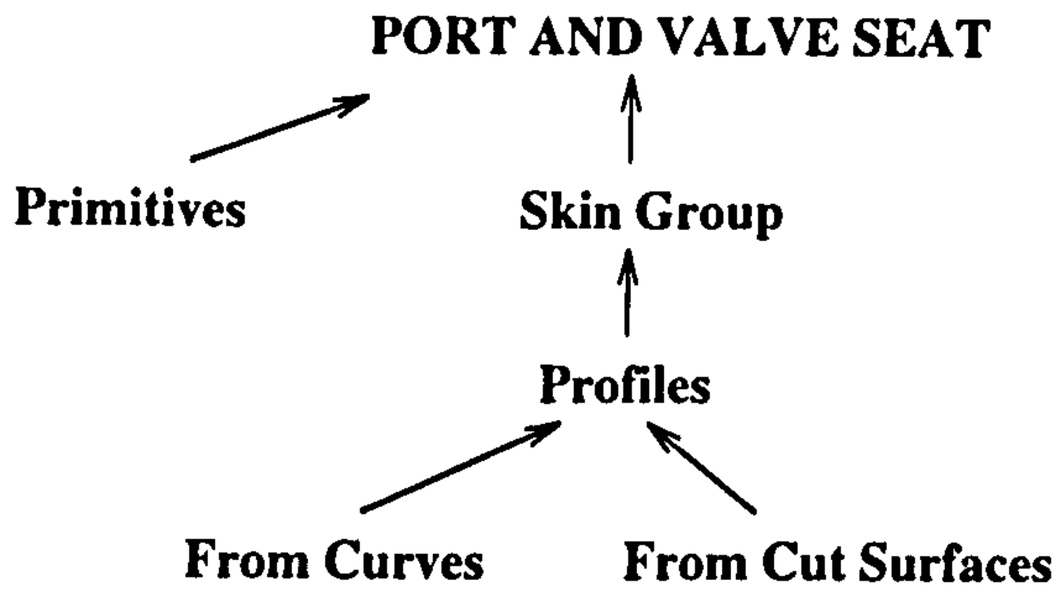


Figure 2.5 Methods for creating a solid port and valve seat

The geometry of the port was too complicated to be created as a solid object from I-DEAS primitives, but could be built up by skinning a set of cross-sectional profiles. The cross-sectional profiles of the port therefore had to be defined first of all.

Profiles could not be generated directly from the original CADDS curves accurately in Level IV. Instead, the CADDS surfaces were read into a new model file and slices through these surfaces were taken using the 'Boolean_cut' operation in SM. All Boolean operations in I-DEAS approximate surfaces as a set of triangular facets. These facets can be operated on as

using simple planar geometry techniques. For example, the Boolean_cut operation involves finding the intersection points between a set of triangular faces and a user-defined cutting plane. For each surface, a profile is fitted through the intersection points found.

Since, say, four CADDS surfaces defined the port walls at one lengthwise position, four profiles were generated where these surfaces intersected the cutting plane. A single profile representing the entire cross-section of the port could then be created by joining together these individual profiles. This procedure was repeated at a number of positions along the length of the port.

The cross-sectional profile of the valve seat was created in a slightly different manner since the valve seat was only modelled in CADDS as a set of curves. However the valve seat was a simple shape with rotational symmetry. A profile of the axial cross-section was therefore entered as a set of points, and this profile was then rotated about its axis to form a solid model of the valve seat. This gave a set of surfaces which could then be cut to create a cross-sectional profile in the same way as the rest of the port.

A distinction must be made here between geometric points and profile points, which are different entities in I-DEAS. A profile may be defined by listing a set of geometric points in order around a closed or open loop. Profile points are then generated in the position of those geometric points and labelled pp(1) to pp(max) where max is the number of geometric points used to define the profile. The labelling of the profile points therefore depends upon the order in which the geometric points defining the profile are selected.

Once defined, the cross-sectional profiles were listed as a skin group and skinned to create a solid model in the following way. Two cross-sectional profiles are considered which lie at different positions along the length of the port. The skinning process creates a curve that follows the profile between profile points labelled pp1 and pp2 on the first profile and creates a similar curve on the second profile. Another curve is created that joins the points labelled pp1 on each profile, and a fourth curve is created by joining the points labelled pp2 on each

profile. A Coons patch surface [51] is then created from these four curves as shown in Figure 2.6. A Coons patch surface is defined only from its bounding curves and so can only create a surface accurately if that surface curves in at most one direction.

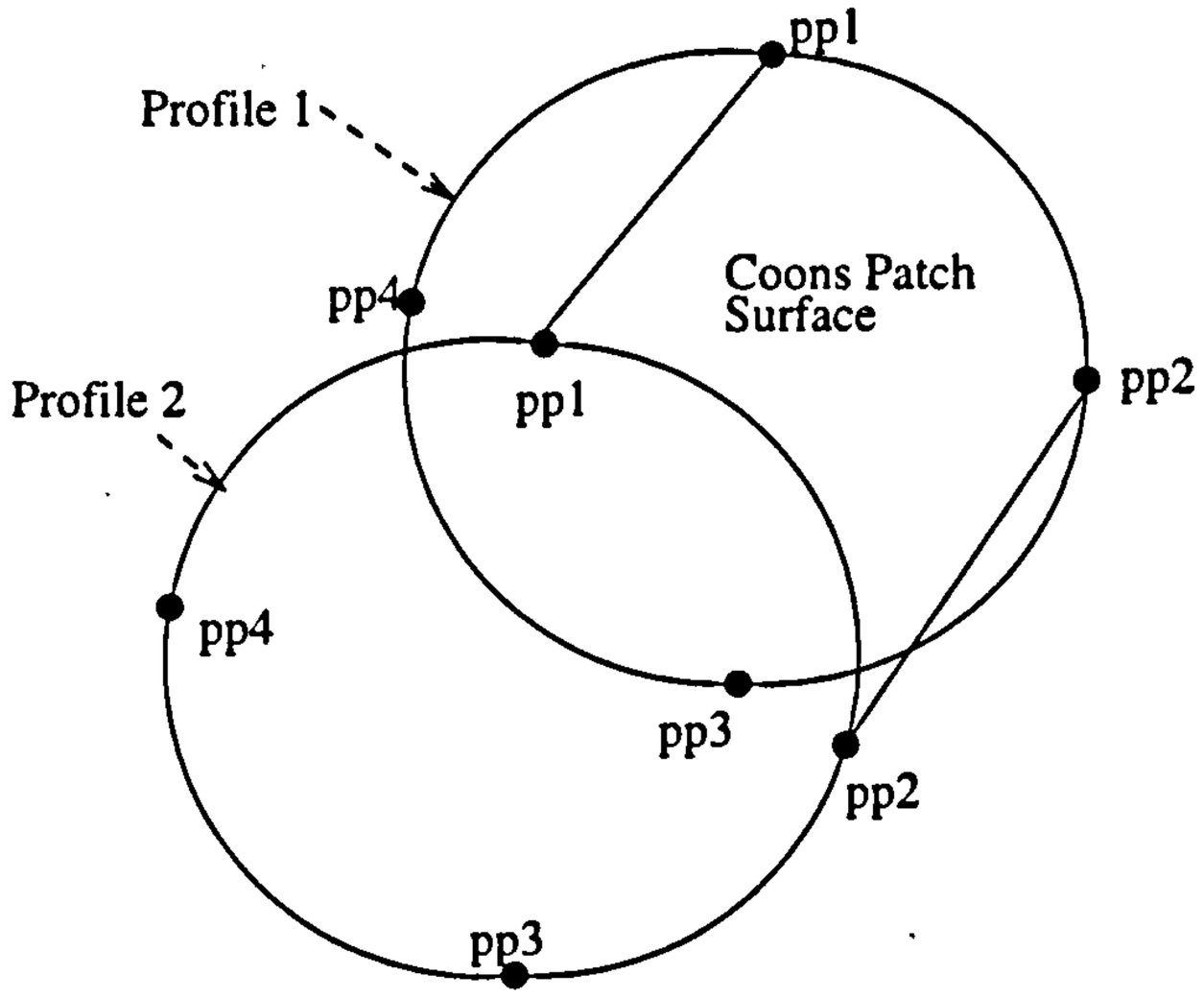


Figure 2.6 Coons patch surface created between two profiles

Points labelled pp2 and pp3 of the two profiles are then taken and a second surface is created as before, and so on around the profiles. It is clear that skinning requires that all of the profiles in the skin group have the same number of points.

Profiles generated from cut surfaces have points created where each facet of the surface intersects the cutting plane, and so every cross-sectional profile does not necessarily contain the same number of profile points. The cross-sectional profiles created from the port surfaces therefore had to be modified to ensure that they all contained the same number of profile points. Profiles could not easily be modified in I-DEAS IV, however, so a new set of cross-sectional profiles was created using the old cross-sectional profiles as templates.

Three profile points in each cross-sectional profile were screen-picked to define a work-plane. A work-plane is a 2D working area that can be defined anywhere in the 3D model, and once set up, any points and curves generated by screen-picking are created to lie on the work-plane. With the work-plane aligned to a cross-sectional profile, geometric points were created by screen-picking positions along that profile. A new profile was then created by listing the newly created geometric points in order. All profiles created in this way contained the same number of points. Also, each geometric point was positioned so that it would be roughly in line, lengthwise, with the corresponding points on adjacent profiles so that the surfaces created are not twisted.

Even when the profile points were aligned, however, the port still displayed some faults in I-DEAS IV. In the skinning process, the curves fitted between profiles can be straight lines or splines. A spline is fitted between the same profile points in a number of profiles, whereas straight lines can be used to join two adjacent profiles. If straight lines were used, then the surfaces created would not follow the smooth curvature of the port as shown in Figure 2.7 and so splines were used in skinning.

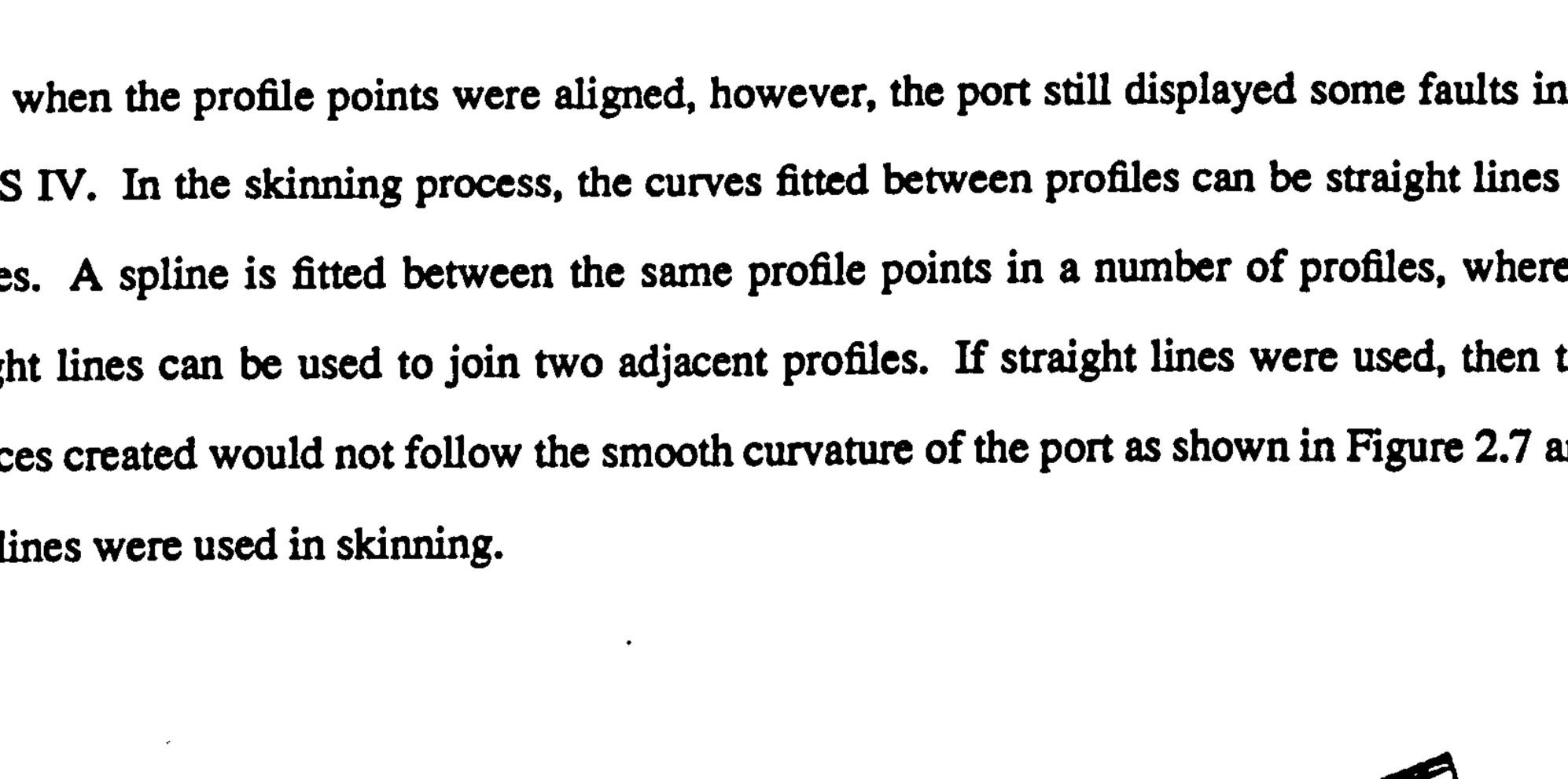


Figure 2.7 Straight lines used in skinning port

There is a tight bend in the port as it approaches the valve, however, and the spline curves fitted between the profiles gave a rippling effect as can be seen in Figure 2.8. This rippling was caused by the spline being fitted through too many profile points. Here a surface created from too many profiles lead to distortion around the tight bend in the intake port. By contrast, too few profiles in a surface would mean that the correct curvature in the port was not followed.

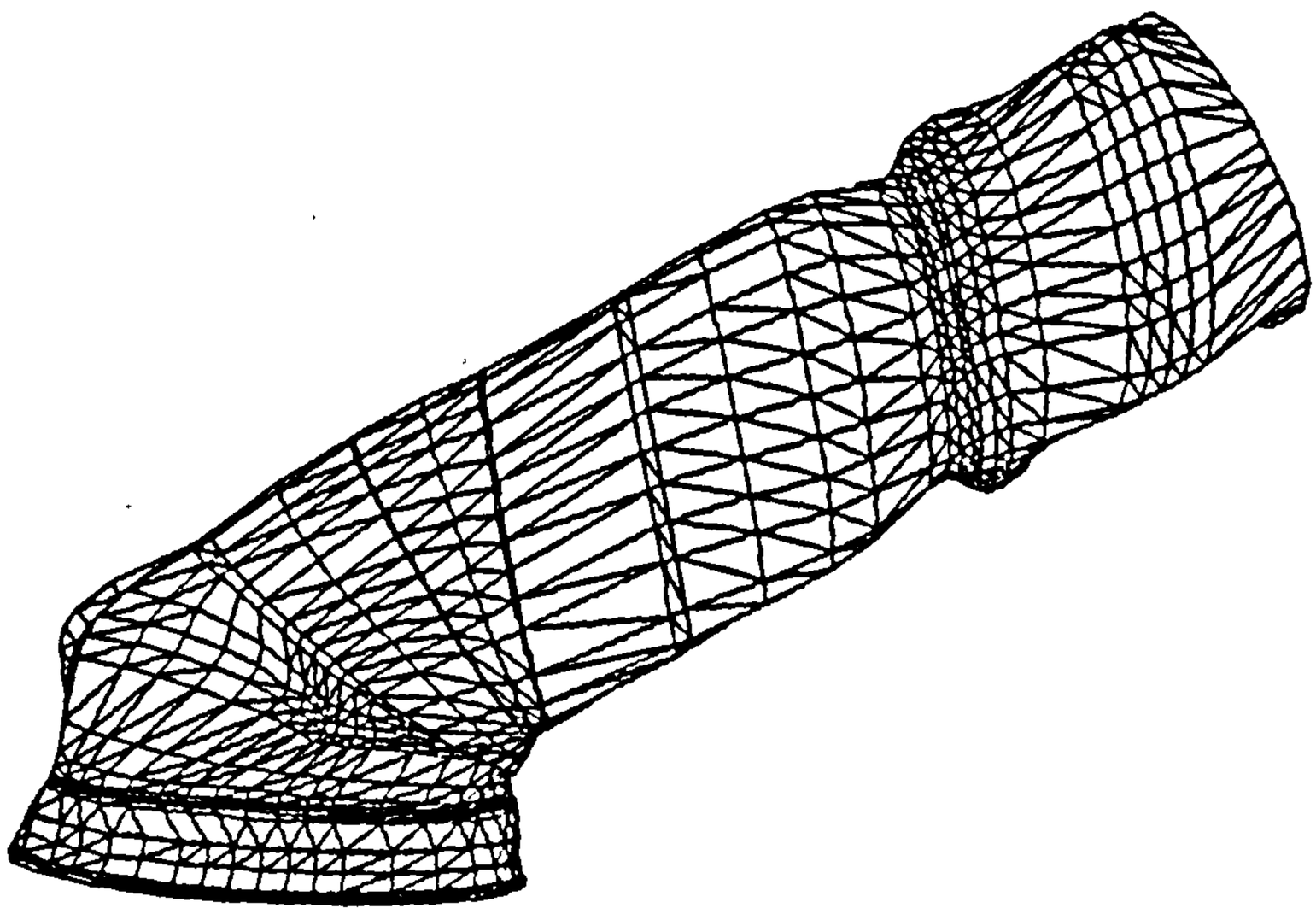


Figure 2.8 Too many profiles used for port geometry

A simplified model of the port was therefore created by using a reduced number of profiles in the skin group. This simplified port was created by skinning seven profiles which was a compromise in the number of profiles required for accurate detail, and the number of profiles to give undistorted surfaces, erring on too few surfaces to ensure that the surface distortion was not too great. The simplified solid port is shown in Figure 2.9 and it could be used to create a complete simplified solid model of the geometry, but first the other regions of the

geometry had to be modelled.

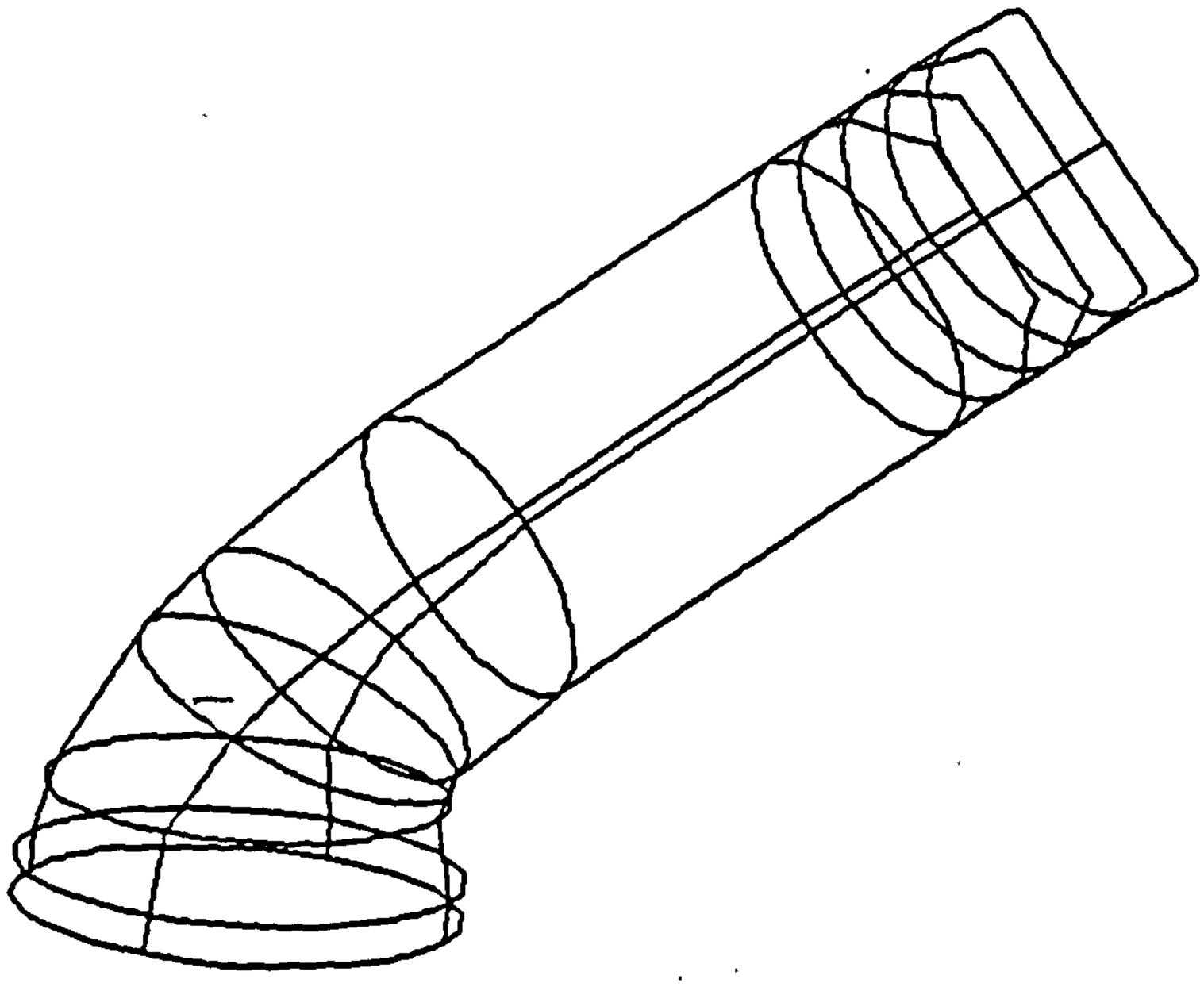


Figure 2.9 Simplified solid model of port

2.6.2 Creating the roof geometry

The pent-roof of the combustion chamber had been created in CADDS as a set of surfaces. Like the port, a solid model of the roof was first created by Boolean cuts of these surfaces to form profiles, and by skinning the resultant profiles. Similar problems of surface distortion arose, however. The techniques used to create a simplified port model were laborious and too a long time. Therefore alternative methods for generating the roof were sought.

The geometry of the roof was found to be suitable for creating a solid model of the roof directly from primitives, and so a cylinder primitive was created with the height and diameter of the roof. This primitive was then cut by two user-defined planes and the appropriate

sections were removed. The resultant solid model is shown in Figure 2.10.

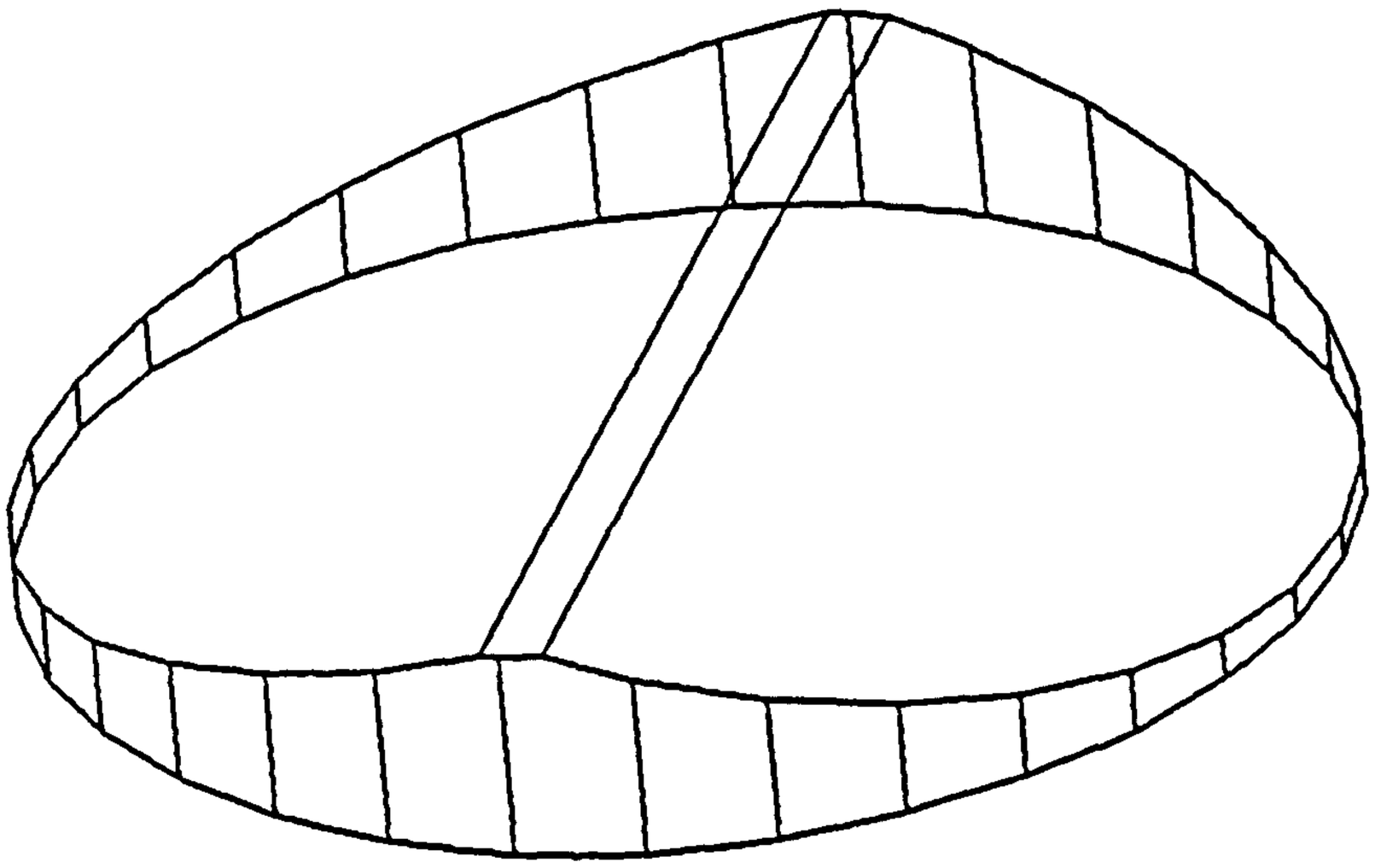


Figure 2.10 Solid model of roof from primitives

From this solid, convex fillets or 'rounds' as they are referred to in I-DEAS were created along the upper surface edges as shown in Figure 2.11.

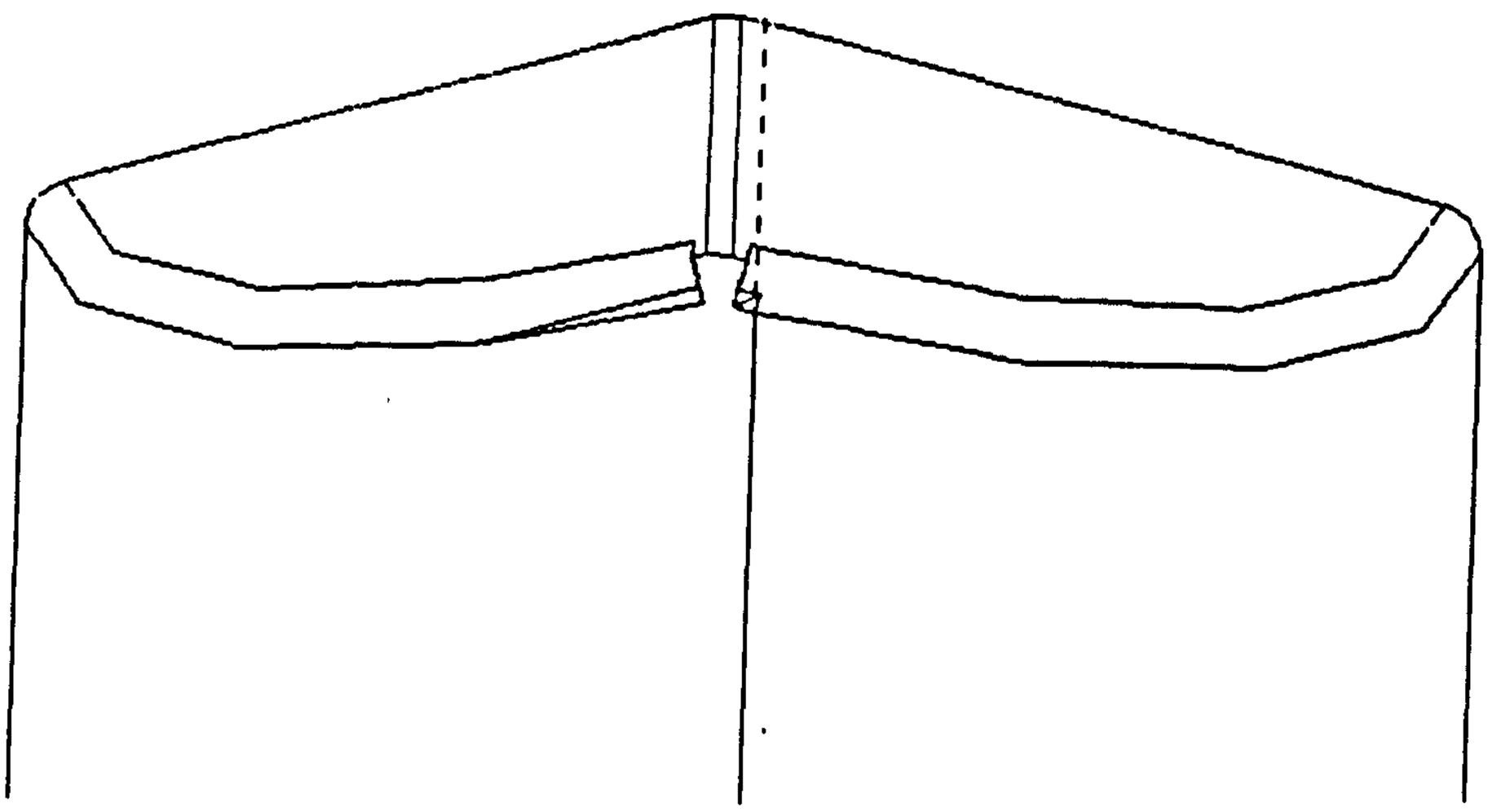


Figure 2.11 Filleted edges on solid model of roof

I-DEAS IV could only create the intersection of two rounds if their original edges met at 90^o, and so the intersection of the rounds at the top of the cylinder roof could not be done in SM. This problem was not resolved in versions V or VI of I-DEAS, and so the filleting was done only for the two longest edges of the roof. Even then this solid model had small surfaces near the ends of the two fillets, and these were too small for meshing in FEM.

Thus a simplified solid model of the roof which contained no filleted edges at all was created along with an alternative solid model with fillets along its longest edges of the roof. During the subsequent mesh generation, both the filleted and simplified models were used with different mesh generation techniques as will be discussed in chapter 3.

Finally, since the symmetry of the geometry meant that only half of the roof had to be modelled, both of the solid models of the roof were cut in half along the cylinder axis.

2.6.3 Creating the valve and valve guide geometry

The geometry of the intake valves and valve guides were supplied as engineering drawings. Their computer models had to be created entirely within I-DEAS and added to the geometry of the port and roof in order to create a complete computer model.

The engineering drawings were used to define 2D cross-sectional profiles of the geometries. Where tolerances had been specified in the engineering drawings, the middle of the tolerance range was chosen. These profiles were created in I-DEAS by entering the coordinates of the profile points and creating straight lines between them. The valve fillets were reconstructed by a simple fillet command and the 2D valve profile is shown in Figure 2.12.

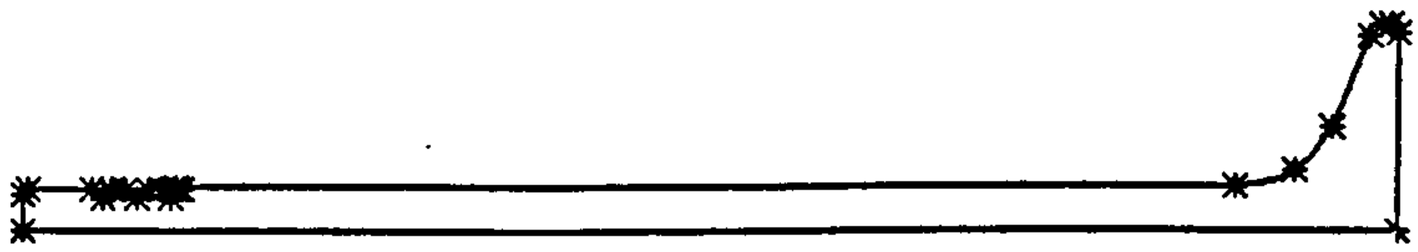


Figure 2.12 Valve profile in 2D

These profiles were then rotated through 360^{O} in SM and the path of the rotated profile points defined a set of curves in 3D. Surfaces are automatically created from these curves to produce a 3D axisymmetric solid model as illustrated in Figure 2.13. The same procedure was also used to create a solid model of the valve guides.

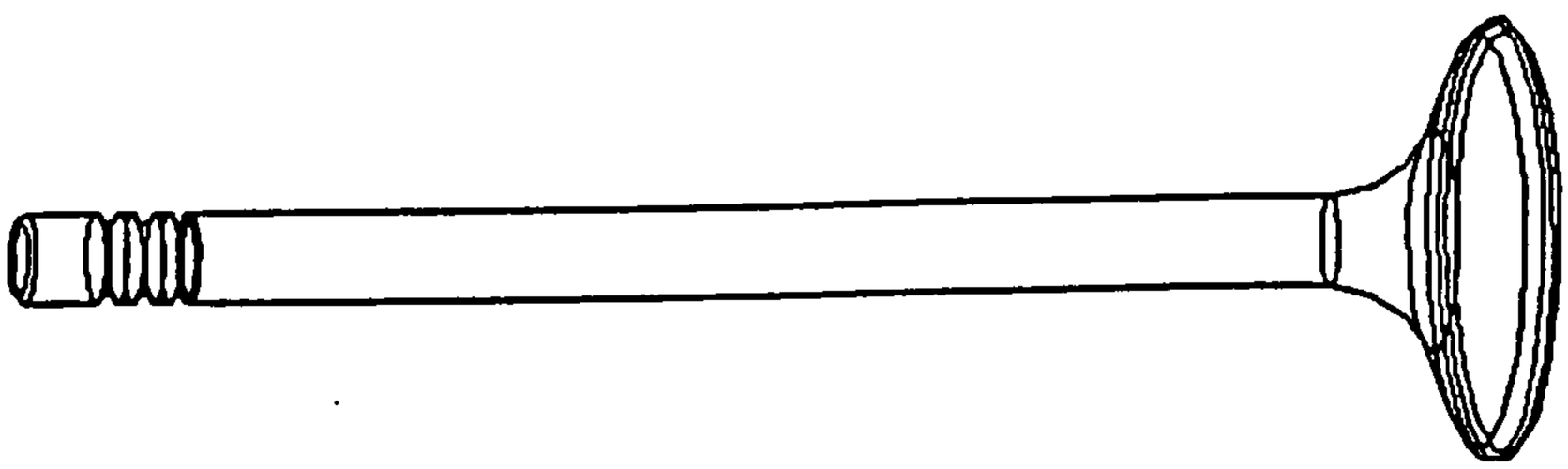


Figure 2.13 Solid model of valve

2.6.4 Creating the cylinder geometry

The cylinder itself could be constructed in two ways. Firstly it could be created as a cylinder primitive and joined to the solid roof using Boolean operations. This approach proved to be

unsuccessful, however, because mis-matches in the facet representations of the roof and the cylinder gave rise to tiny surfaces being created at their intersection. In order to solve this problem, the height of the cylinder was simply added to that of the primitive cylinder used to create the roof. The roof was then created as described in section 2.6.2 before.

2.7 Constructing the complete solid model

Solid models of the port and valve seat, roof and cylinder, valve and valve guide had been created, but these models were not joined together. The Boolean_join command in SM was therefore used in order to join the port and the roof, and from this the valve and valve guide were to be cut, using Boolean operations once more.

2.7.1 Joining the port to the roof and cylinder

Boolean operations, using faceted representations of the surfaces to be joined, use tolerance values to determine whether two facets intersect or not. Thus if two surfaces touch one another in their solid model form, their faceted forms may be found to touch only in some places and not in others due to the tolerances used. This may produce gaps and create tiny internal facets between the two joined solids which could not then be meshed.

Therefore the solid models of the port and the cylinder had to fully intersect one another before they could be joined using the Boolean operations. In order to create such an intersection, the simplified port was extended by copying the valve seat profile a distance of 2mm along the valve axis to a position within the roof region. This new profile was added to the skin group and the simplified solid model of the port was then recreated. The solid port then intersected the roof fully and the two could be joined.

2.7.2 Cutting the valve from the model

The valve and valve guides are obstacles in the flow and so must be holes in the solid model

of the flow domain. Thus the valve and valve guide were aligned with the port and cylinder, and cut from the joined solids using Boolean operations.

The solid objects modelling the valve and valve guide were read from the model file in which they were created, into PEARL, and then into the model file containing the port-roof-cylinder geometry in SM. Objects read in to a model file retain the same orientation to the coordinate axes and origin as those used when they were created. Thus the valve and valve guide had to be moved from their original position to the correct position with respect to the cylinder roof.

The method used was to displace each object a set distance along its axis of symmetry, then to rotate it about two axes into correct alignment, and finally to translate the object by moving it's origin to a reference point such that it sits in the correct final position as illustrated in Figure 2.14. The coordinates of this reference point were determined in FEM, where the centreline of each valve is displayed, as can be seen in Figure 2.1. The reference point lay at the intersection of the centreline and the pent-roof of the combustion chamber, and was found by listing the coordinates of that point in I-DEAS.

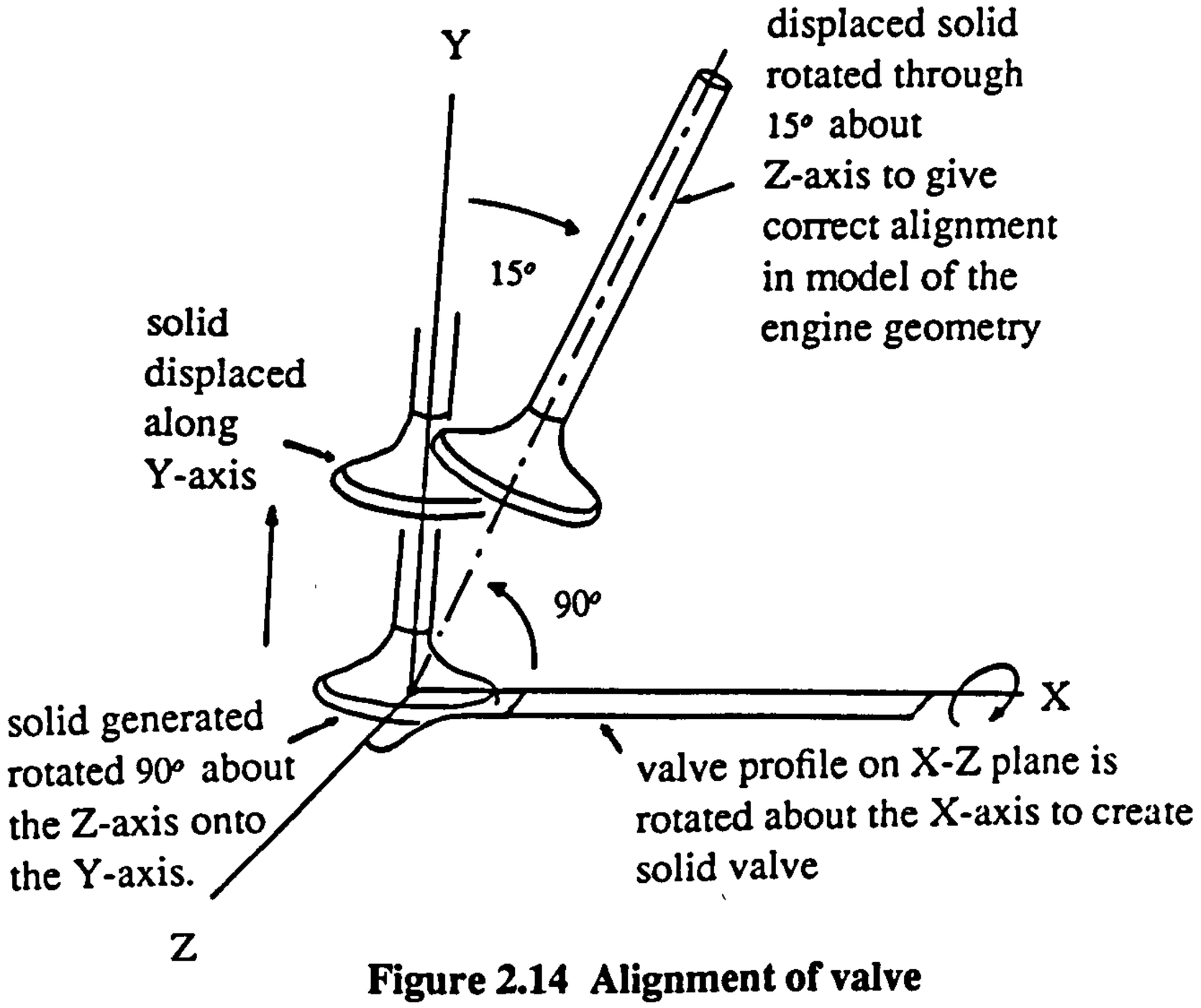


Figure 2.14 Alignment of valve

The distance from the reference point to each object along the centreline of the valve was obtained from the engineering drawings, and this measurement was then the set displacement of the object along its symmetric axis.

The process was speeded because the valve and valve guide have the same centreline. The angle of rotation required was determined directly from the engineering drawings to be 15^O about one axis. The objects then needed a rotation of 90^O about another axis to be aligned. A valve lift was added given to the valve by displacing it along its centreline.

In I-DEAS V, a new capability allowed objects to be automatically aligned to any user specified face or centreline. Therefore the aligning of the valve and valve guide became a much simpler process. Once correctly aligned in SM, the valve and valve guide solid models were cut from the port and cylinder model, so creating a valve and valve guide shaped 'hole' in the solid model. This completes the port-valve-cylinder solid model.

2.8 Creating a computer model from curves

A third option for creating a coherent computer model of the engine geometry from the fragmented CADDS data was identified in Section 2.5. Creating the model would involve modifying the CADDS curves first and then creating Coons patch surfaces from these curves. It was recognised that this option may well take considerably longer than using the solid models described in the preceding Sections. The computer model created in this manner will not be a solid model, but rather a coherent surface model. This means that the mesh generation method must be using mesh volumes, as will be described later in chapter 3. It is sufficient to say here that the structure of the surfaces in this model must necessarily be carefully constructed with the mesh generation in mind.

Once the surface structure has been planned, the method for creating surfaces in I-DEAS is very simple. Four curves must be created first, which lie on the surface of the geometry. These four curves are screen-picked in turn and a Coons patch surface is generated between

them. The nature of the Coons patch surfaces, as described previously in Section 2.6.1, means that highly curved regions of the geometry must be divided into a number of individual surfaces. A requirement of I-DEAS is that adjacent surfaces share only one, entire curve between them. This restriction means that the structure of the mesh must be planned with these considerations in mind also.

The model created in this work, called model-3, contained a port created from CADDS curves and Coons patch surfaces as described above. However the curves describing the roof, valve and cylinder were not created based upon CADDS curves but rather they were based upon the wireframe geometry derived from the solid models already created. For the roof, this wireframe description was simpler than the original CADDS curves whilst for the valve and cylinder the CADDS curves did not exist at all.

2.9 Models created

Based upon the three options available for creating a model of the geometry from the CADDS data, three models were created as illustrated in Table 2.1 below.

Model	Port	Valve	Roof	Cylinder
Original data	CADDS curves and surfaces	CAD drawings	CADDS curves and surfaces	Specifications
Model-1 [solid]	Simplified solid	Solid from profiles	Simplified solid	Solid
Model-2 [solid]	Simplified solid	Solid from profiles	Simplified solid + fillets	Solid
Model-3 [wireframe geometry]	Curves built from CADDS	Curves from solid	Curves from simplified solid + fillets	Curves from solid

Table 2.1 Computer models created in I-DEAS

Table 2.1 first lists the origin of the data for each of the components in the engine geometry, as was mentioned in Section 2.3. Model-1 is a solid model created using boolean joining and cutting from the solid components as described in Sections 2.6 and 2.7. The simplified port is used, see Section 2.6.1, along with a simplified roof described in Section 2.6.2. Figure 2.15 shows model-1. Model-2 is also a solid model created in the same way as model-1, except that a filleted version of the roof was used, as shown in Figure 2.16. Model-3, by contrast, was created using curves. Here the port was constructed from CADDS curves, whilst the rest of the components were first created as solids and then the underlying wireframe geometry was extracted to give a set of curves. The roof of model-3 was created from the filleted version of the solid roof. Each of these models were used to generate meshes, as will be described in the following chapter.

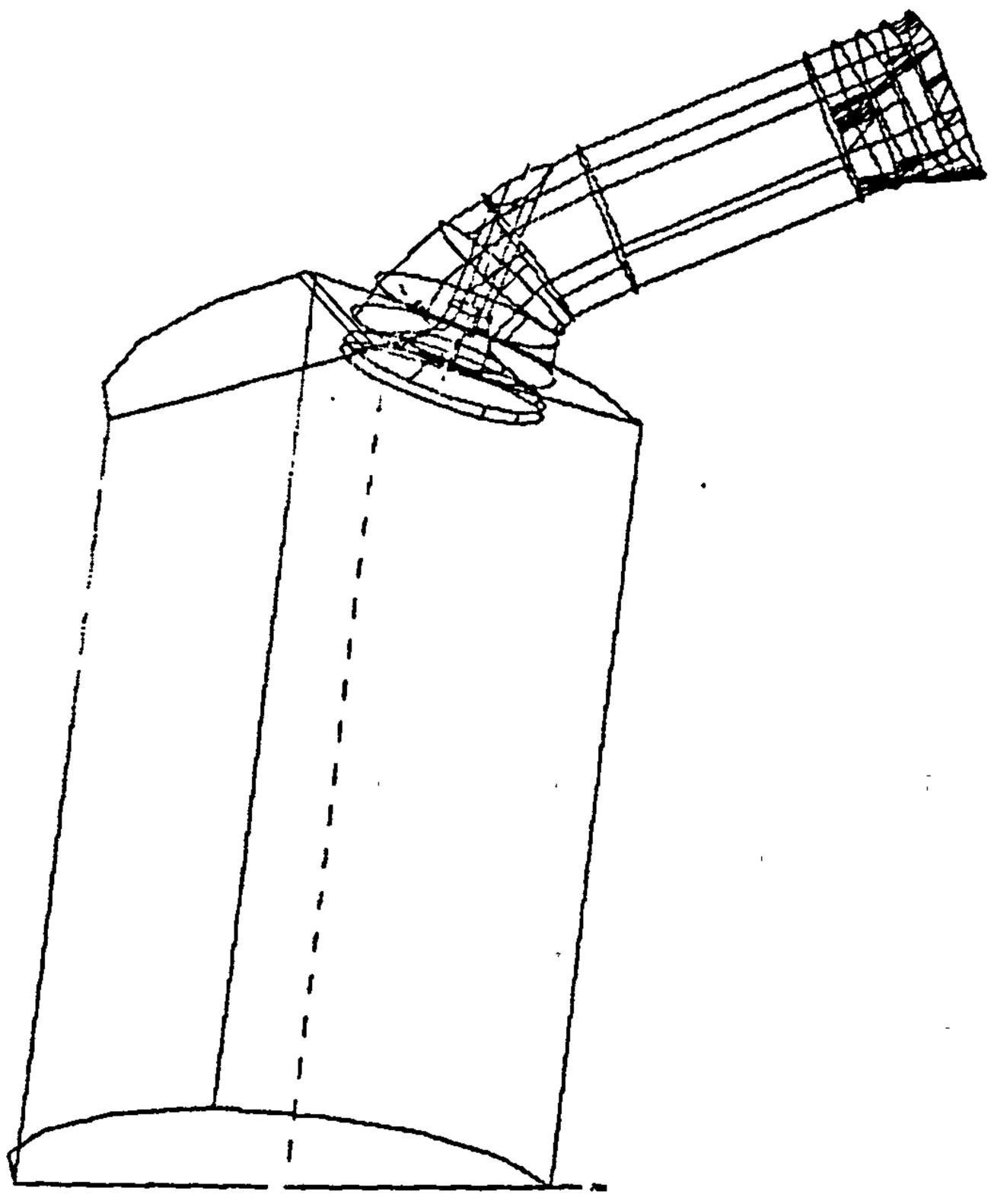


Figure 2.15 Model-1 of port and cylinder geometry

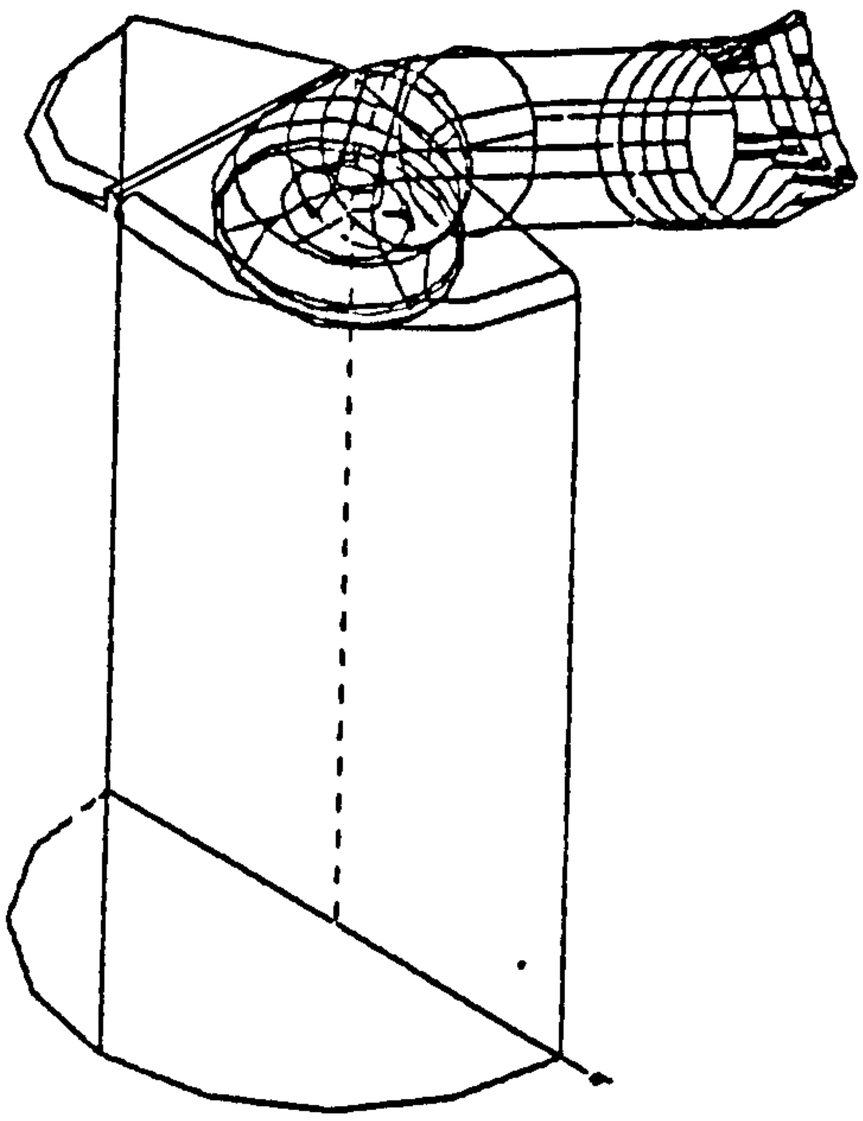


Figure 2.16 Model-2 of port and cylinder geometry

2.10 Implications for Jaguar

A number of methods of model creation have been examined in this chapter. It is clear that there is no simple route from a CADDS description of the engine geometry to a model of the geometry that is suitable for meshing. Reffering back to Figure 2.2, route C or D may be used depending upon the application. Solid models are fast to generate in SM, following route D, and the entire geometry can be modelled in one to two weeks by a skilled I-DEAS user. The drawback of solid models is that they cannot always represent the geometry accurately. On the other hand, the methods of building points and curves in FEM (route C) is slower, taking about three weeks to create an entire model, although the model created can be a better representation of the real geometry than the solid models since less simplifications are made.

Two options are therefore open to Jaguar, which include;

1. Retraining the CAD draughtsmen to create models suitable for mesh generation.
2. Using the I-DEAS package for all aspects of CAD and mesh generation.

Using I-DEAS throughout the company may not be an acceptable policy, tying the company to one CAE package when new developments are continually occurring for CAE applications. This could mean that the best CAE packages are not being used for different tasks in the company. Retraining a number of draughtsmen to include considerations of analysts would technically be a sound move, since it would also benefit the FE structural analysis that is routinely done in Jaguar. This option may simply be too costly, however.

2.11 Summary of model generation

Methods for reading CAD data into I-DEAS and generating solid models in I-DEAS have been explored in this chapter. Whilst the IGES data should be read into PEARL initially, route B in Figure 2.2, no one route was found to be suitable for all aspects of the geometry. Therefore simplifications to the pent-roof and an approximation of the intake port geometry have had to be made. Advantage was also taken of a symmetry plane in the geometry, and so the solid models only needed to define half of the geometry. The test of these methods depended upon their usefulness in generating high quality computational meshes, and so judgement of which route was most suitable was not made until meshes were generated.

Three models were created in order to test the geometry handling methods available in I-DEAS. These models must also be assessed for their suitability to be meshed easily. The next stage, therefore, is to generate meshes using these models, and this is described in the next chapter.

3. GENERATING A MESH FOR THE ENGINE

3.1 Introduction

The computer models of the engine geometry described in chapter 2 now need to be meshed using the I-DEAS FEM module. Whilst FEM does not have a one-step mesh generator for creating hexahedral cells, the module does have methods for generating meshes that involve some degree of automation.

The general techniques for mesh generation in FEM are first described in Section 3.2. Section 3.3 outlines the different meshes that were generated using the two methods available in I-DEAS. Details of these meshing techniques, as applied to the computer models described in chapter 2 are given in Sections 3.5 and 3.6. The quality of the mesh and its influence on mesh generation and solution is then discussed, followed by a description of methods for modifying the valve lift for a given mesh. A summary of the chapter is then given in Section 3.9.

3.2 Mesh Construction in I-DEAS

Meshes are generated in FEM by dividing the geometry into a set of volumes and then filling each volume with vertices and cells. These volumes may be meshed individually using a structured mesh, referred to in I-DEAS as a'mapped mesh'. Alternatively, an unstructured or 'free mesh' may be used in which the number and distribution of cells is not constrained in the volume.

The technique for generating a mesh within the volume involves defining a 'mesh area' on each face of the volume, then grouping these mesh areas to form a closed 'mesh volume'. The size, distribution and number of cells to be generated in the mesh volume is defined by the user. From this information, the appropriate vertices and cells are then generated automatically in FEM. Quasi-two-dimensional cells, that is, 2D cells distributed over a

surface in 3D co-ordinate space, are first generated on the mesh areas. These cells are used as the basis for generating the 3D cells within the mesh volumes. When the full 3D mesh has been generated, FEM then deletes the quasi-2D cells.

Adjacent mesh volumes share a common mesh area to ensure that the mesh is continuous across the boundaries between mesh volumes. Thus one mesh is created throughout the geometry, built up from a set of individual meshes within the mesh volumes.

3.2.1 Mesh Areas

A mesh area is a region defined on a surface upon which quasi-2D cells can be generated. The mesh area is created as follows. Curves are selected in turn until a closed loop has been defined. These curves form the boundaries of the mesh area. A single FEM command will automatically search the geometry for suitable loops of curves and so the user can quickly define the required mesh areas.

If the curves defining the mesh area all lie on one surface, the mesh area and its underlying surface are associated. If no such surface is found, a Coons patch surface will be fitted to the curves defining the mesh area. Since a Coons patch surface is defined only from its bounding curves, it can only create a surface accurately where the surface curves in at most one direction. If the required surface curves in two directions, interior points on the surface are needed to make the surface definition more accurate, and so one Coons patch is unlikely to produce the desired surface. Where the original surface curvature is too great, then, the user should divide the surface into a set of smaller Coons patch surfaces. Since these smaller surfaces will have less curvature than the original surface, then that larger surface will be described more accurately.

3.2.2 Mesh Volumes

Mesh volumes are created by defining a set of mesh areas which form the boundaries of a

closed volume. Again automatic searches in FEM can speed up this selection process.

Mapped mesh volumes are limited to six mesh areas, whereas free mesh volumes could be made up of a maximum of 50 mesh areas in I-DEAS IV and up to 200 in subsequent releases. This means that a complex geometry may be meshed using fewer free mesh volumes than mapped mesh volumes, and consequently free mesh volumes are generally quicker and easier to create.

Mapped mesh volumes can only contain hexahedral cells, whereas the cells in free mesh volumes could be tetrahedral or hexahedral in I-DEAS IV. In I-DEAS V, however, free meshes were restricted to tetrahedral cells. This reduced functionality in I-DEAS was because the hexahedral cells generated automatically in I-DEAS IV were not of good enough quality to be reliable for FE stress analysis, which is the usual application for I-DEAS meshes. The operation was therefore removed in Level V due to considerations of quality for the I-DEAS package as a whole.

3.2.3 Mesh Checking

Before the vertices and cells are generated in a mesh volume, some pre-meshing checks should be made by the user. If there is excessive mesh area curvature, the mesh generated may not follow the geometry closely. Also, the user must check for the existence of any 'free boundaries' or 'holes' within the mesh volume, since a free boundary will cause discontinuities in the mesh if it lies inside a mesh volume.

Once the mesh checking has shown that an acceptable mesh will be generated, the mesh generation itself, that is, the assignment of vertex and cell labels and the placement of vertices, is done automatically in FEM.

3.3 Meshes generated

The models described in the previous chapter were used to generate three meshes, which are listed in Table 3.1. Mesh-1 was created from model-1 which is a solid model. Free mesh volumes were created and meshed using the automatic hexahedral mesh generator in I-DEAS IV. Mesh-2 was also created using model-1, but this time tetrahedral cells were automatically generated using I-DEAS V. These tetrahedra were then subdivided into hexahedral cells. The third mesh, mesh-3, was created from model-3 using mapped mesh areas in which hexahedral cells were generated. The valve in this mesh was then moved to produce three valve lifts.

Mesh	Model	Mesh volumes	Cells
Mesh-1	Model-1	Free	Hexahedra
Mesh-2	Model-1	Free	Tetrahedra split into hexahedra
Mesh-3	Model-3	Mapped	Hexahedra

Table 3.1 Meshes generated

Mesh-1 and mesh-2 are described in detail in Section 3.4, and mesh-3 is described in Section 3.5. A discussion of mesh quality is then given in Section 3.6 and the valve movement used for mesh-3 is described in Section 3.7.

3.4 Generating the Engine Mesh Using Free Meshes

Two methods of mesh generation using free mesh volumes were explored for the engine geometry, using hexahedral cells in I-DEAS IV and tetrahedral cells converted to hexahedra

in I-DEAS V, creating mesh-1 and mesh-2 respectively.

Mesh-1 was a hexahedral free mesh generated in I-DEAS IV using techniques of mesh area and mesh volume definition. The whole engine geometry in model-1 had too many surfaces to create one single free mesh volume and so mesh areas were created inside the geometry to divide the solid model into a number of mesh volumes. Having defined the mesh volumes for the unstructured hexahedral mesh, the cell size needs to be defined. Free mesh volumes require only a global cell size in order to create the mesh. This global cell size may be different for each mesh volume, however, which gives the user some control over the mesh density.

A coarse mesh was required in the first instance in order to test the mesh using a minimum of computer resources and time. A global cell size of 2mm was the maximum that could be successfully generated around the valve, however, due to its small details and high curvature. Thus in the roof region, about 20,000 2mm sized cells were generated. A similar number were generated throughout the port, although the port was split into two mesh volumes with global cell sizes of 4mm and 6mm. The combustion chamber has 40,000 6mm sized cells throughout, which gave mesh-1 about 80,000 cells over the whole geometry.

However an algorithmic restriction in STAR-CD v2004 limited the number of faces that could be associated with any one vertex in the mesh. Although this restriction was removed in the next version of STAR-CD, I-DEAS V had been released by then and no longer supported free meshes with hexahedral cells. Thus mesh-1 could not be used in this work and another method had to be devised in order to generate free meshes using hexahedral cells.

In I-DEAS V, a one step command could create free mesh areas and free mesh volumes directly from a solid object. This facility was used to generate mesh-2, which was a free mesh with tetrahedral cells created from model-1. Mesh-2 was a very coarse tetrahedral mesh containing only 2321 cells so that the technique of cell subdivision could be tested quickly and using little computing resources. Each tetrahedron in mesh-2 was therefore

subdivided into four hexahedra using a 'tetra-to-brick' mesh conversion program written by the author which is outlined as follows.

A tetrahedron can be subdivided into four hexahedra by creating new points at the midpoints of its edges, centroids of its faces and at the centroid of its volume. These points are then joined up to form four new hexahedral cells, as shown in Figure 3.1.

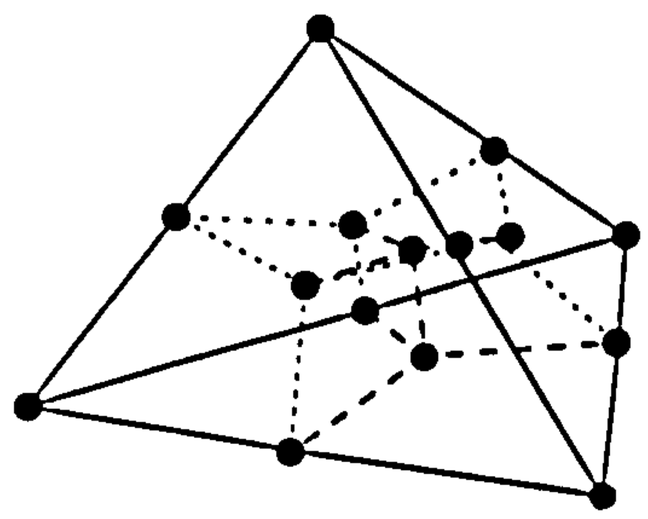


Figure 3.1 Splitting a tetrahedron into hexahedra

In the tetra-to-brick program, each tetrahedron is divided in turn. This means that two adjacent tetrahedra will have new vertices generated in the same positions along their mutual faces and edges. These duplicate vertices must be merged in order to create a coherent mesh. PROSTAR has a fast vertex-merging command and so the hexahedral mesh was written out from the program as STAR format vertices and cells, and read in to PROSTAR. After vertex merging, the result was a completely unstructured, coarse hexahedral mesh of the flow domain containing 9284 cells.

3.5 Generating the Engine Mesh Using Mapped Mesh Volumes

In this method of mesh generation, model-3 was used, which was a wireframe geometry as is described in Section 2.8. As mentioned in that Section, the curves and surfaces created were

highly dependent upon the mesh structure. Thus the details of the mesh structure are presented in this Section. The wireframe geometry consisted of points and curves only, which were then used to create mesh areas and mesh volumes.

An important consideration for building the mapped meshes is that each mapped mesh volume must be a topological brick, and so holes cannot be readily included into the surface of the geometry being meshed. To overcome this restriction, the stem of the valve was cut short, so that the valve ended inside the port instead of cutting the port surface, and the valve guide was omitted completely as can be seen in Figure 3.2.

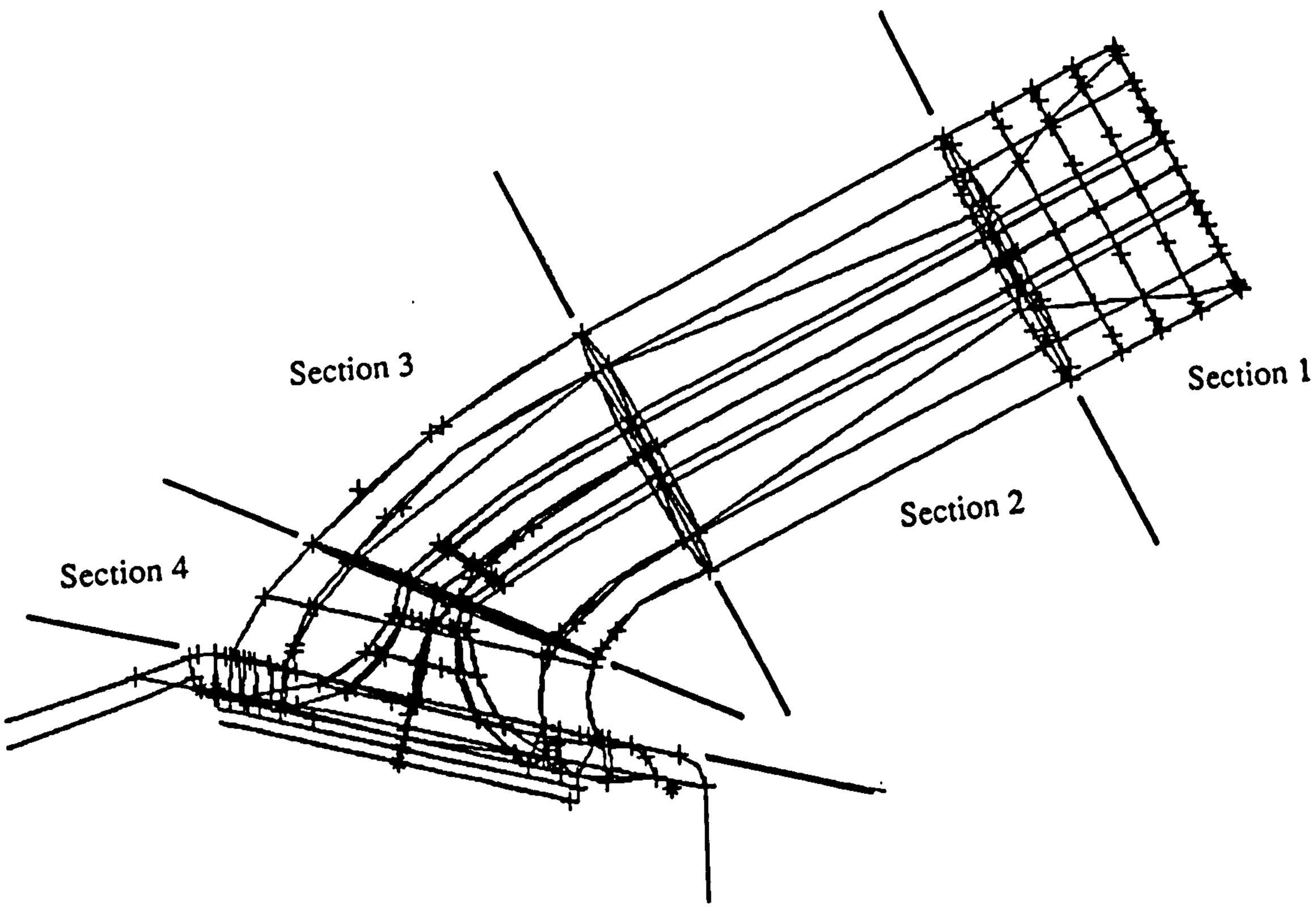


Figure 3.2 Mesh structure with cut valve stem

This follows from the work of Luo and Bray [52,53] whose results showed little difference

between the in-cylinder flow field and turbulence levels generated by a port with a truncated valve stem and a port with a complete valve stem. They concluded that the valve may be confidently modelled with a truncated valve stem and so this approach was used in this work when generating mapped meshes.

3.5.1 Building Basic Meshes

The process of creating mesh areas and mesh volumes in FEM has been described in Section 3.2. For mapped meshes, the cell size, distribution and quality is strongly linked to the structure of the mesh volumes. The creation of mesh volumes must therefore be carefully planned throughout the flow domain. Here the geometry has been divided into three regions, the port, the valve and the roof and cylinder.

3.5.2 Creating the Port and Valve Seat Structure

From the intake manifold, the intake port which is oval in cross-section splits into two separate intake ports which have circular cross-sections as illustrated in Figure 3.3.

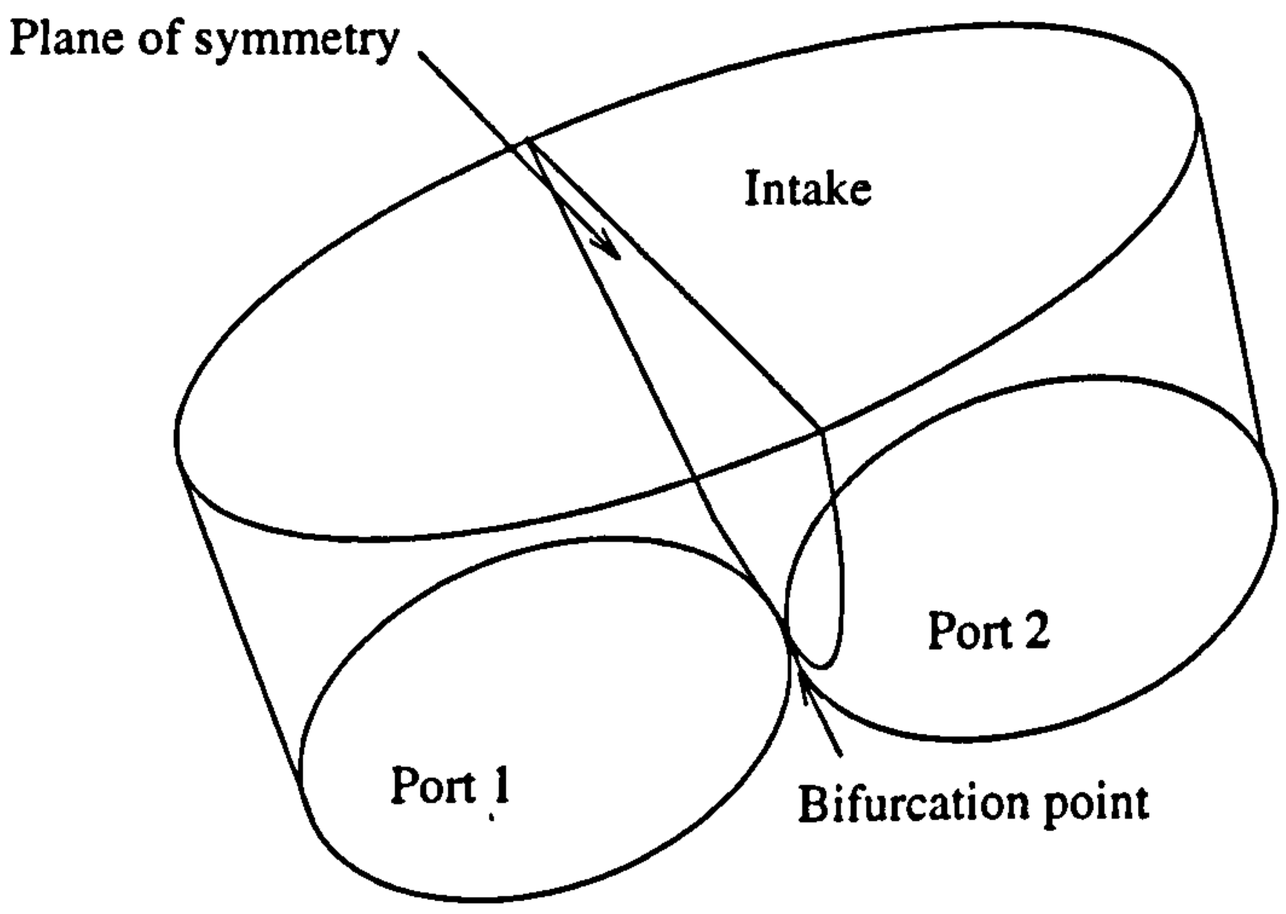


Figure 3.3 Bifurcation of the intake port geometry

The engines symmetry plane splits the intake geometry into two and so the geometry to be meshed has a D-shaped cross-section which becomes circular downstream, as illustrated in Figure 3.4a).

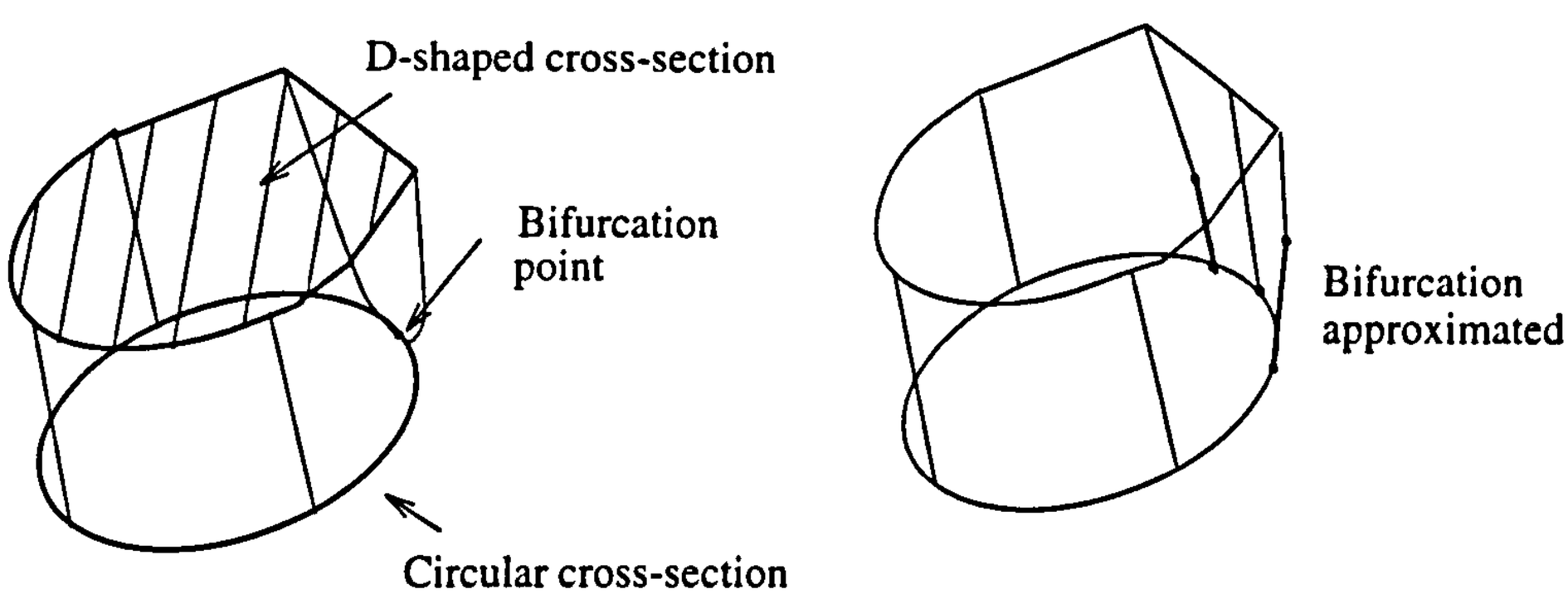


Figure 3.4 Cross-sections on one intake port

From this Figure, it can be seen that the straight line defining the flat of the 'D' shrinks to a single bifurcation point and the cross-section becomes circular. This is not a viable structure for a mapped mesh volume in I-DEAS. An approximation to this geometry therefore had to be made, as illustrated in Figure 3.4 b).

Once the geometry had been modified, thirty two mesh volumes were created. As shown in Figure 3.2, four sections were created lengthwise along the port and eight mesh volumes were created per section. The structure chosen for the mesh volumes accounted for the truncated valve lying in the port.

3.5.3 Creating the Roof and Cylinder Structure

The wireframe geometry of the roof and cylinder included the longest edges at the top of the roof filleted with a 3mm radius fillet as illustrated in Figure 2.11. Since the filleting operation

in I-DEAS V could not fillet curves that intersected at angles other than right angles to one another, the intersection of the two fillets had to be modelled by hand. This manual filleting involved deleting some curves and creating a new set of curves that merged the two fillets together smoothly. Four curves were then used to create a Coons patch surface between the two fillets, bridging the small gap between them.

Once the geometry had been modified in FEM, the roof and cylinder geometry required further subdivision in order to create a set of mapped mesh volumes. Since the eventual mesh was to be continuous, the structure of the mesh volumes in the port and around the valve stem had to be incorporated into that of the roof and cylinder and continued past the valve itself situated within the roof.

3.5.4 Meshing the Port, Valve and Cylinder Structures

Having created the mesh volume structure, the cell distribution must be defined before I-DEAS automatically generates the vertices and cells in each mesh volume. Cell size and distribution is defined by the number of cells along each curve of a mapped mesh volume. The nature of mapped mesh volumes propagates cell densities throughout the mesh. The initial cell sizes were therefore selected after examining a region of the geometry in which the mesh density was critical. In this case, the critical region lies between the valve and the wall of the port.

Since the wall-functions used in STAR-CD approximate the flow in cells adjacent to mesh walls, there must be sufficient cells between the walls to be able to calculate the flow field. The sufficient condition depends upon the flow regime and can be determined by using successively finer meshes until the results are independent of the mesh. Typically this means a mesh with at least 10 cells between the walls. Since cell densities are propagated throughout the mesh very large overall mesh sizes can be obtained by using small cell sizes. Indeed port and cylinder meshes generally have around 200,000 cells or more. The computer resources were limited in this work, however, and so a coarser mesh density had to be used. Therefore,

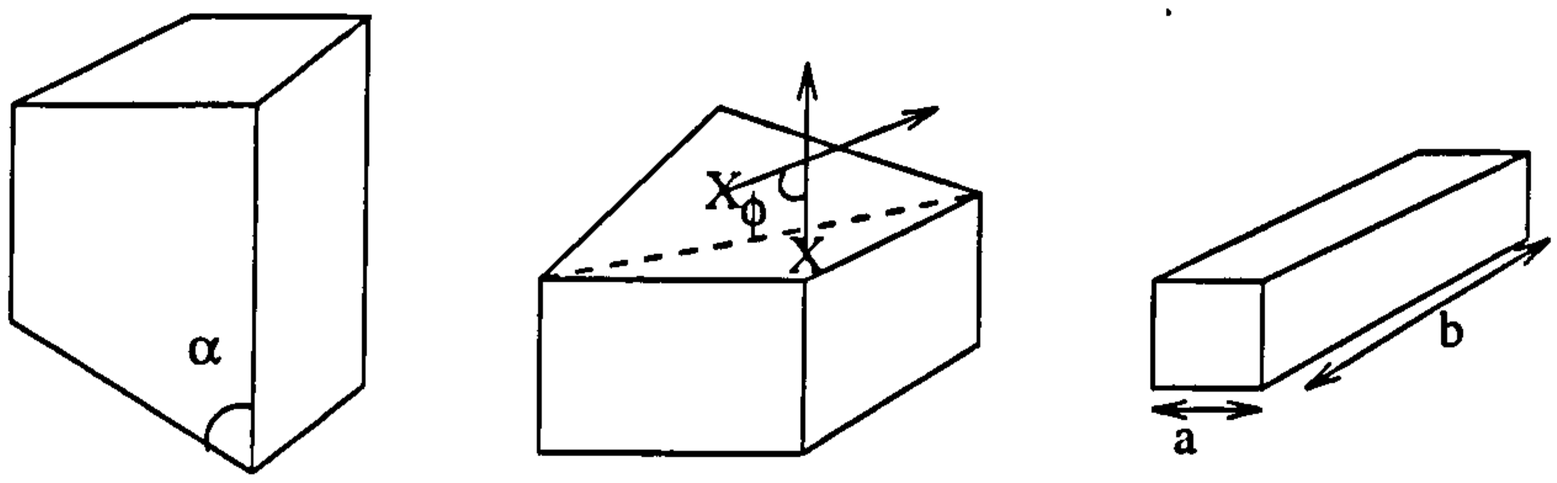
six cells were placed between the walls of the port and the surface of the valve. This mesh density sets the length of one side of a set of cells, and the other cell side lengths were chosen to keep the shape of the hexahedral cells as close to a brick shape as possible. Thus mesh-3 was a mesh of the whole port and cylinder geometry with a truncated valve stem, containing around 92,000 hexahedral cells.

Mesh-3 was generated with a 6mm valve lift and two other valve lift cases of 4mm and 8mm were also required for CFD analysis of steady flows in the port and cylinder. New meshes for these valve lifts could have been created by creating new models with different valve lifts, but this would have taken a long time. An alternative is to distort the existing mesh near the intake valve. Before moving the valve is described, however, concepts of mesh distortion and mesh quality should be discussed.

3.6 Mesh Quality

Throughout the mesh generation process, care must be taken to ensure that the mesh is of 'good quality'. Mesh quality includes the distortion of individual cells, their distribution throughout the mesh, and the size of the cells.

The distortion of a cell is its deviation from an ideal cell shape. In the case of a hexahedral cell, the ideal shape is a 'brick', where the longest edge of the brick follows the direction of the general flow. Thus all the angles on every face of the ideal cell are 90^0, and every quadrilateral face is planar. STAR-CD refers to these measures of cell distortion as internal angle and face warpage respectively, where face warpage is the angle between two triangles created from the four points of a quadrilateral face. A final measure of cell distortion is the aspect ratio which is the ratio of the lengths of the sides of the cell. Figure 3.5 illustrates these three criteria for measuring cell distortion.



internal angle warp angle b/a = aspect ratio

Figure 3.5 Criteria for measuring cell distortion

The amount of cell distortion acceptable to a CFD code is determined by the type of discretisation of the flow domain and the assumptions made by the code in this discretisation, and the solution algorithm used, which have been described in Section 1.5.1, and also by the flow field itself. STAR-CD is a finite volume code in which flow derivatives are calculated at cell faces from the difference of the flow variables at the cell centres. If internal angles are high then the derivatives calculated for one face of the cell may correspond to a different part of the flow domain, as is demonstrated in Figure 3.6.

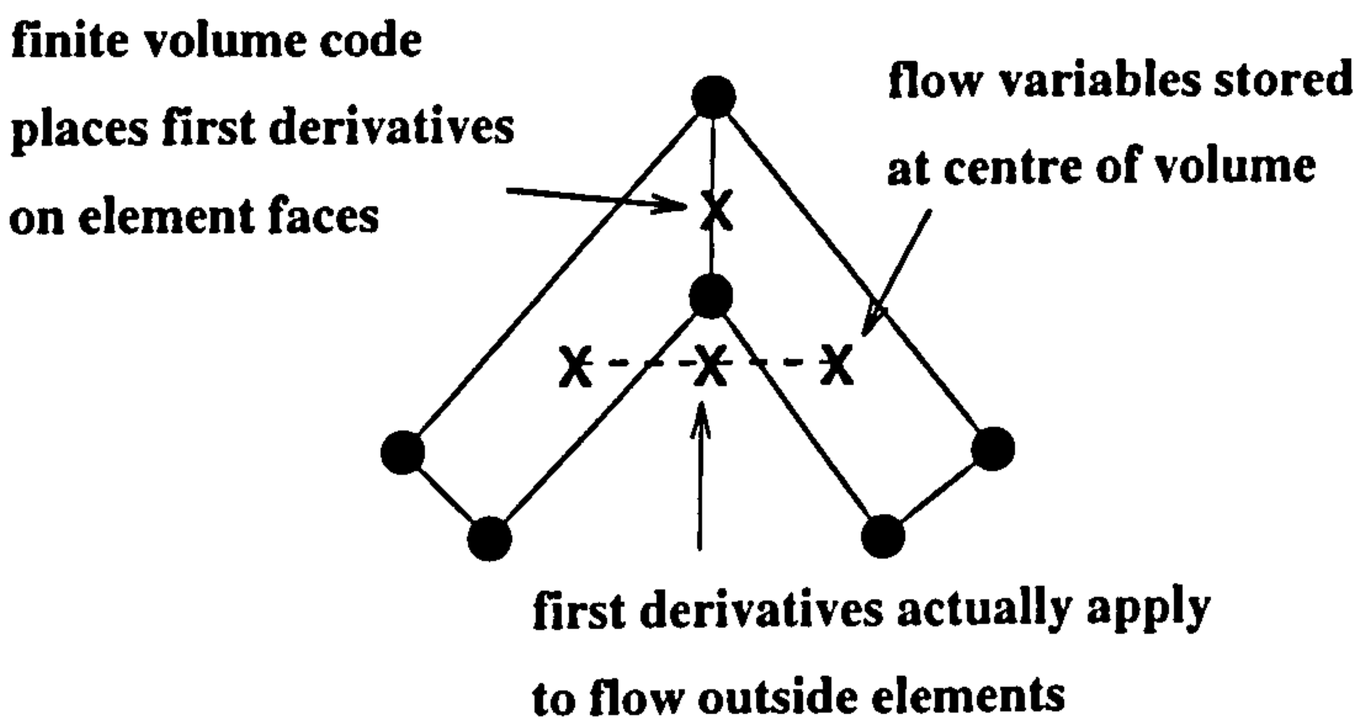


Figure 3.6 Cell distortion effecting calculation of flow variables

Thus excessive cell distortion can result in the flow field being incorrectly calculated. Table 3.2 shows the recommended maxima for cell distortion for using STAR-CD, although if a cell has combinations of cell distortion, then these values may be too high. In practice, however, the results are also influenced by the flow field, and in regions where the flow has low gradients the cell distortion which will still give acceptable results can be much higher.

Aspect Ratio	Internal Angle	Warp Angle
10	45 degrees	45 degrees

Table 3.2 Maximum cell distortion recommended for STAR-CD

Another indication of mesh quality is the size of the cells and their distribution throughout the mesh. Since the flow variables are assumed to be constant over each cell, flow features that are smaller than this mesh size cannot be determined. Therefore an ideal mesh is one for which the solution is 'grid independent', that is, a smaller mesh size would not change the solution over the flow domain. For many engineering flows, however, a grid independent solution would require an excessively large number of cells which would take too long to analyse, and so the cost would be prohibitive in industry. Therefore meshes are sized to pick up the salient flow features without needing to calculate every last flow detail.

The quality of the mesh, therefore must be considered with respect to the physics of the flow itself, the CFD solver user, the computational resources available and the amount of detail required in the solution. This is still then dependent upon the experience and judgement of the CFD analyst. Automatic mesh refinement or 'adaptive meshing' are being researched, for example Dannelogue and Tanguy [54], Moulkalled and Acharya [55], and Evans et al [56]. In these techniques, a coarse mesh is first analysed and then refined locally where the results

show rapid changes in the flow variables. The solution is then found for this new mesh and the solution-adaption loop is continued until an acceptable solution is reached. Such a tool is already available in a recently released commercial finite volume code, RAMPANT [57], which uses a mesh with only tetrahedral cells.

3.7 Moving The Valve

Two methods for moving the position of the valve were attempted, both of which were based on mesh-3 with a 6mm valve lift. The first technique moved vertex positions around the valve using user-defined groups whilst the second technique moved the vertices automatically.

3.7.1 Moving the Valve Manually

The valve in the original mesh is modelled by a gap or hole inside the mesh. Using PROSTAR, the vertices lying on the valve were grouped and written to an external file listing the vertex labels and 3D co-ordinates. This file was then an input for a program written by the author which displaced the vertices by a set distance and wrote them to another external file in the STAR-CD format. To move the vertices along the valve axis required co-ordinate displacements of

$$dx = A\cos\theta \tag{3.1}$$

$$dy = 0 \tag{3.2}$$

$$dz = A\sin\theta \tag{3.3}$$

where A is the displacement along the valve axis, and θ is the angle at which the valve lies to the z-axis in the xz-plane in degrees. For this geometry, $\theta = 15^0$.

There are limits on the distance that a vertex may be displaced within a mesh, however. A

vertex must not be moved to a position where it passes through a cell wall as shown in Figure 3.7, a condition which STAR-CD refers to as a negative volume,

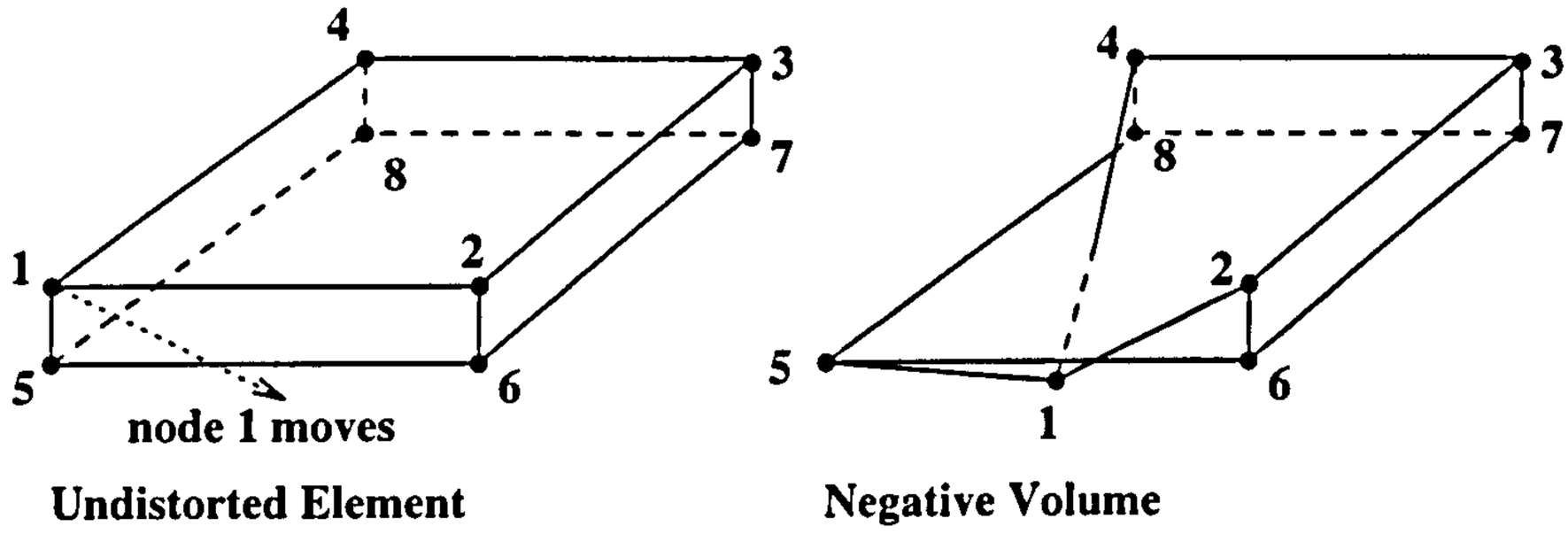


Figure 3.7 Negative volume

The limit of vertex movement is determined by the size of the cells that are attached to this vertex. In this case, a limit of 0.3mm displacement was found for increasing the valve lift, and 0.15mm for decreasing the valve lift. The new vertex coordinates were read into PROS-TAR and Figure 3.8 illustrates the vertex layers that are displaced.

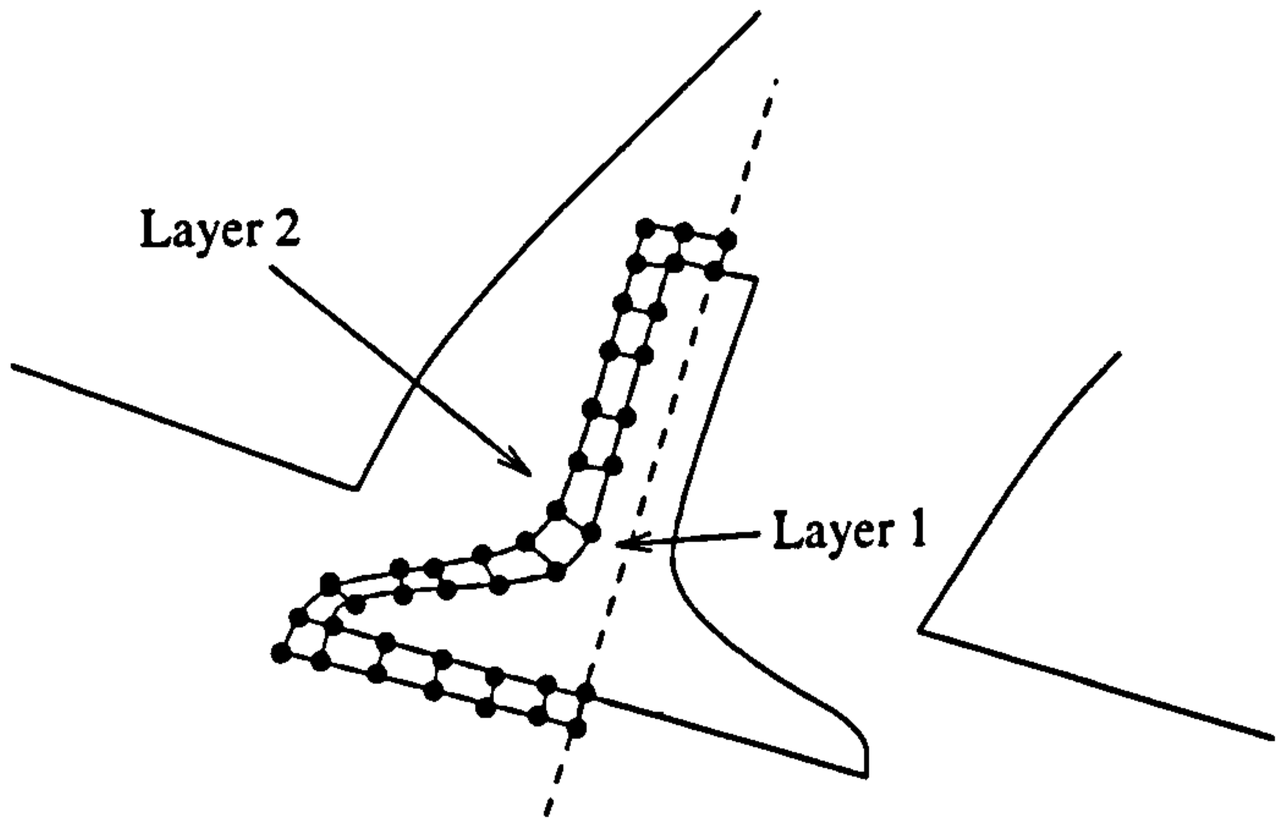


Figure 3.8 Layers of vertices to be moved

For example, layer 1 in Figure 3.8 lies on the surface of the valve, and this layer is first moved by 0.3mm. Next the cells attached to these vertices are listed and all their constituent vertices are stored in a new vertex group. This vertex group consists of layers 1 and 2 in Figure 3.8.

This larger vertex group is then displaced by 0.3mm in the same way as before. Note that the vertices within the group are all moved by the same amount, and so their positions relative to each other are unchanged. Subsequent vertex displacements are made in the same way, finding all the cells attached to the previous vertex group, and forming a new group from the vertices in the associated cells.

When applied to the actual mesh, this technique gave a total valve displacement of 1.5mm and hence a valve lift of 7.5mm. The valve could not be displaced further, however, because any further layers of cells selected would have contained vertices that lay on the surface of the mesh and the actual surface of the mesh would be distorted if these vertices were moved.

The same procedure was used to move the valve to a smaller valve lift in the mesh. Starting from a 6mm valve lift in mesh-3, each successive vertex group was moved 0.15mm, giving a total displacement of 0.75mm, and so creating a mesh with a 5.25mm valve lift. It must be noted that the meshes for the 7.5mm and 5.25mm valve lift cases represent the maximum distortion that the cells around the valve could accommodate, and larger and smaller valve lifts could not be obtained using this method for moving the valve within the mesh. For both these cases, a few vertices had to be moved by hand when they created cells that were highly distorted, with internal angles from 1.4^o up to 178.4^o. Although these cells were not actually negative volumes, the distortion was still so great that STAR could not determine if they were negative volumes or not.

If a smaller valve displacement had been applied to the vertices, less distortion would have occurred, and no manual 'tweaking' of the mesh would have been necessary. If a mesh was required for a valve lift outside the 5.25mm - 7.5mm range using this method, however, then

a new mesh would have to be built from I-DEAS.

3.7.2 Moving the Valve Automatically

Another, more automatic method of moving the valve from a 6mm valve lift was also attempted. This method moves the valve and its surrounding vertices automatically. The method takes each vertex in the mesh in turn, and finds the nearest vertex that lies on the outer surface of the mesh, and the nearest vertex that lies on the surface of the valve. The chosen vertex is then moved along a vector which is the path of the valve, moving a distance proportional to its relative distance to the non-moving surface and valve surface vertices just found, according to the following equation;

$$X_{new} = X_{old} + [d_2/(d_1+d_2)] * R \qquad (3.4)$$

where

X_{old} = old x,y,z co-ordinates

X_{new} = new x,y,z co-ordinates

R = movement vector of valve

d_1 = distance to nearest valve vertex

d_2 = distance to nearest wall vertex

Thus a vertex lying on the valve surface will move the whole distance of the valve lift, whilst a vertex on the wall of the mesh will not move at all. All the other vertices move in way a sympathetic to this. Figure 3.9 shows examples of vertices moved using this valve-moving algorithm.

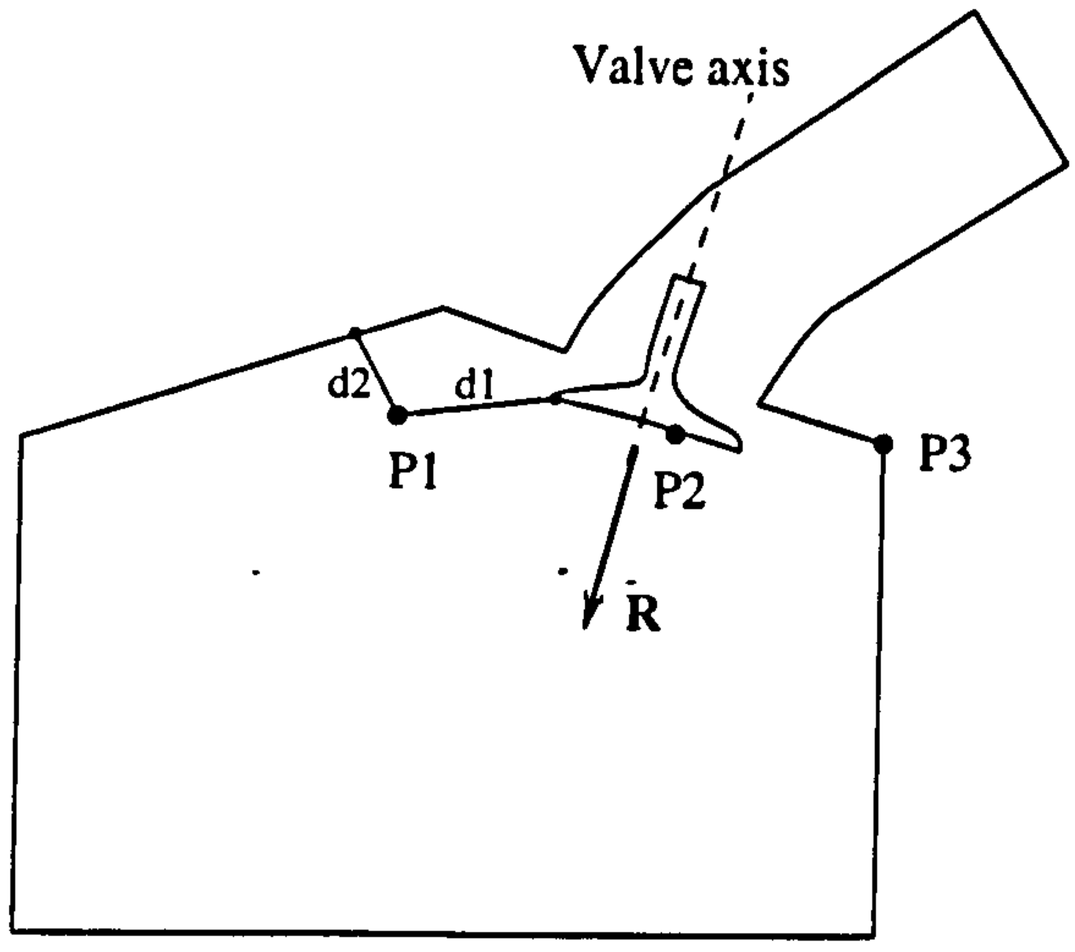


Figure 3.9 Vertices moved using automatic valve moving algorithm

In Figure 3.9, vertex P1 lies inside the mesh with its distance to the nearest valve vertex, d1 and its distance to the nearest wall vertex, d2. Vertex P2 lies on the valve, so that d1 = 0. Thus the vertex moves the full extent of R. Vertex P3 lies on the wall and so d2 = 0 which means that P3 does not move.

Three files were written in PROSTAR which contained, respectively, the vertices on the valve surface, the vertices on the walls of the mesh, and all the vertices in the mesh excluding the vertices on the walls. The program read in these three files and calculated the movement for each valve, and wrote out a new file containing all the vertices in the mesh with their new co-ordinates. This file was read into PROSTAR where the original mesh was stored, and the new vertex positions overwrote the old vertex positions as they were read in.

Using this automatic method for moving the valve, slightly greater valve lifts were achieved, although some cell distortion did occur. Thus some of the most distorted cells had to altered by hand in PROSTAR to improve the quality of the mesh until each mesh would be acceptable by STAR for analysis.

A new mesh, mesh-4, was created in the course work in order to make the mesh more representative of the experimental rig, as will be described in more detail in Section 4.4.3. This new mesh was rebuilt in I-DEAS based upon model-3 with a 6mm valve lift, but was designed for valve movement from the start. Using the automatic vertex-moving program, meshes with valve lifts of 4mm and 8mm were created from mesh-4. Figure 3.10 shows the cell distortion of the 4mm, 6mm and 8mm valve lift meshes that was eventually acceptable to STAR for the analysis runs.

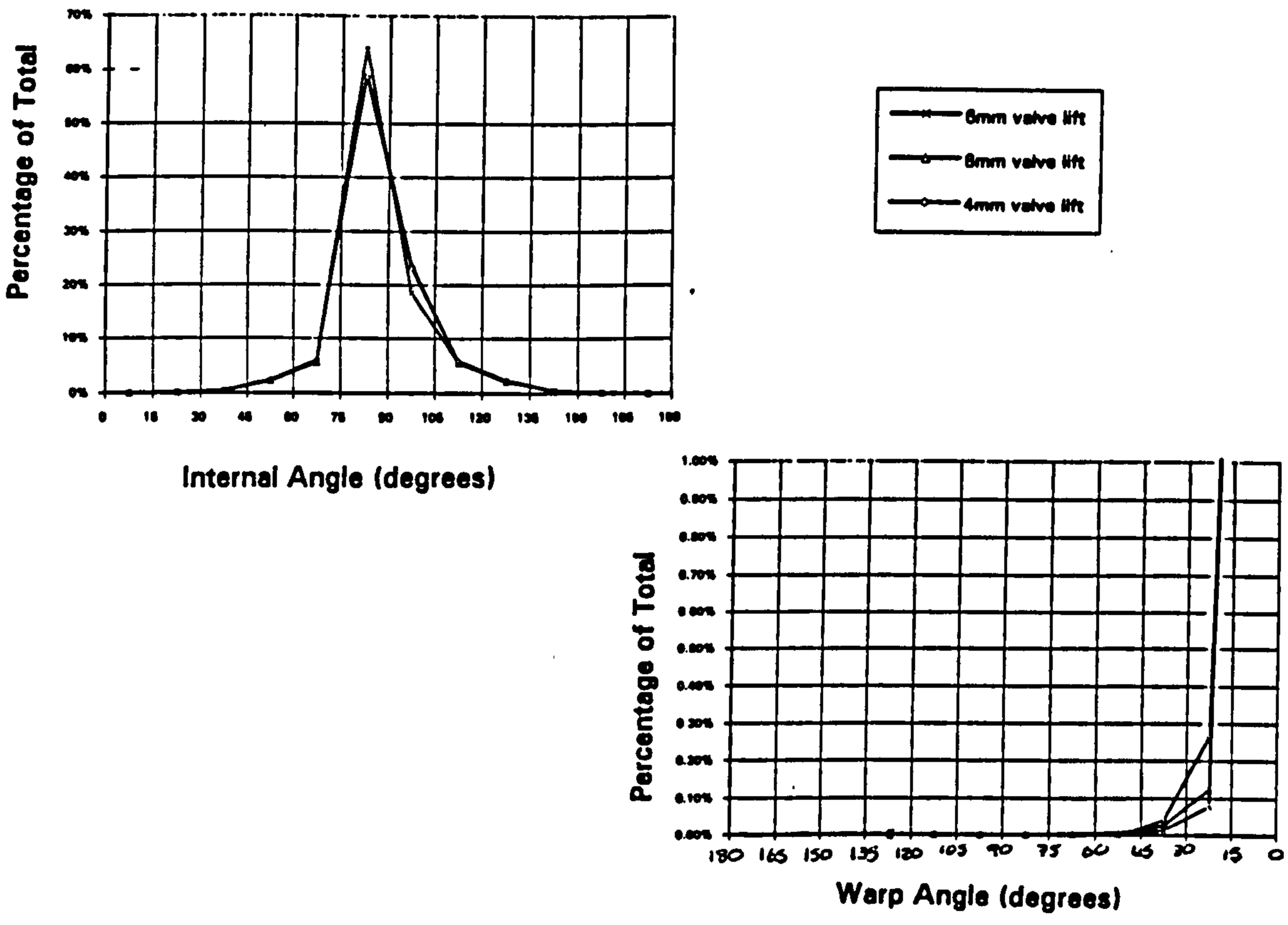


Figure 3.10 Cell distortion for 4mm, 6mm and 8mm valve lift cases for mesh-4

3.8 Quality of the meshes used in this work

The quality of the meshes generated in this work varied considerably, and had a significant influence upon which meshes were suitable for CFD analysis.

As mentioned in Section 3.4, the automatic hexahedral mesh generation used for mesh-1 was removed from later versions of I-DEAS, effectively making that technique obsolete.

Mesh-2 contained cells created by splitting tetrahedral cells into hexahedra. This produced cells with internal angles greater than 179.3^o. Mesh-3 contained cells of better quality than mesh-2. Indeed the mesh quality in mapped meshes is determined by the mesh structure used, and so the mesh quality depends upon the skill and experience of the user as well as the geometry to be meshed. Mesh-4 had slightly better cell quality than mesh-3, and the cell distortion values for the three valve lift cases of mesh-4 are shown in Figure 3.10. Whilst the maximum internal angle shown is high, the most distorted cells lay in regions of the cylinder with small flow gradients, and high internal angles were not found in the same cells that had high cell face warpage. Figure 3.10 also shows that the cell distortion was more sensitive to warp angle for which the maximum was 129^o than for internal angle, for which the maximum was 175^o.

Both manual and automatic methods for moving the valves had produced some highly distorted cells near the valves, with internal angles from 1.4^o to 178.4^o. Manual movement of individual vertices reduced these angles and the cell quality for each valve lift in mesh-4 is given in Figure 3.10. The cells near the valve had to be of better quality than the 'worst cases' acceptable to STAR-CD because these cells lay in the jet flow region of the geometry where the gradients of flow variables are high.

Thus mesh-2, mesh-3 and mesh-4 were suitable for running using STAR-CD, but the quality of the results that they produced must be ascertained before any recommendations could be made to Jaguar.

3.9 Summary of Mesh Generation

In conclusion, different methods for creating meshes for the engine geometry were explored. The mapped mesh volumes proved to be most suitable for this work, even though the user

effort was greater than that using free mesh volumes. Techniques for modifying the 6mm valve lift mesh to obtain meshes with different valve lifts were also examined, and a vertex moving program was written to do so quickly and effectively. The automatic valve moving program was faster and easier to use than its manual counterpart, but both methods required some modifications by the user to remove excessively distorted cells.

The next step in the CFD process is to perform CFD analysis using mesh-2, mesh-3 and mesh-4. If the meshes are suitable for running at their default 6mm valve lift, then alternative valve lifts will also be studied. These results will also be compared to experimental data, which will be explored in the following chapter.

4.0 CFD ANALYSIS

4.1 Introduction

Once the meshes have been generated, the next step is to define the complete computational model and then run the CFD analysis code. The computational model includes the mesh, material properties of the fluid such as density and viscosity, and a definition of the various boundary regions of the geometry. Flow conditions are then specified for these boundary regions, called the boundary conditions, along with the flow conditions that will be applied to the cells inside the mesh at the start of the calculations, which are called the initial conditions. The computational model is then completed by defining the other numerical parameters relevant to the CFD code being used and to the flow being analysed.

In order to establish a suitable mesh generation method for CFD in engine design, the meshes described in Chapter 3 were imported into STAR-CD. Computational models were defined for mesh-2, mesh-3 and mesh-4 and the analysis performed. The results of these analyses were then compared to experimental LDV measurements taken at Imperial College, London. This will establish whether the meshes are of sufficient quality to produce results that correlate well with the experimental data.

The capabilities and limitations of STAR-CD for creating computational models and for performing the actual CFD analysis runs are described in Section 4.2. Section 4.3 then outlines how the computational models are completed for steady flows. The analysis runs performed for steady flows are described in Section 4.4 along with the comparison of the results obtained to experimental data. A discussion of these comparisons follows in Section 4.5. The creation of the computational model for transient flows is described in Section 4.6 and 2D transient simulations are described in Section 4.7. The implications of using CFD in engine design by Jaguar are discussed in Section 4.8, and finally a summary of this Chapter is presented in Section 4.9.

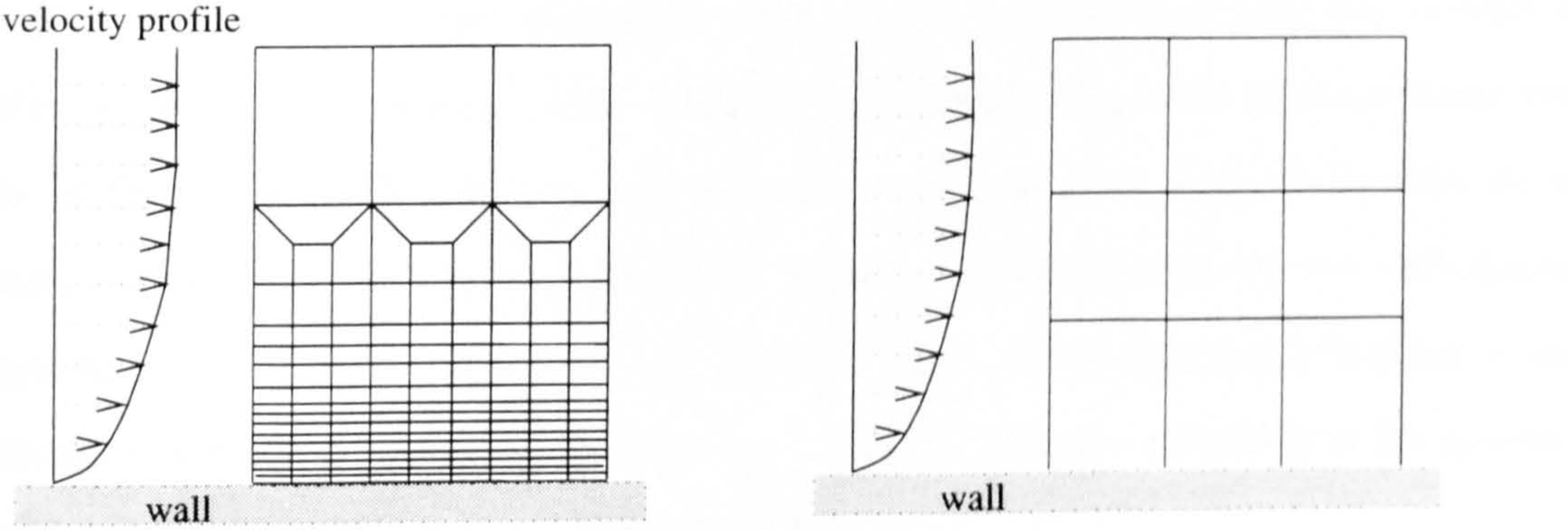
4.2 STAR-CD capabilities for model creation

STAR-CD has been described generally in Section 2.2.1, but here the use of this code is described in detail for creating the computational model and performing the actual analysis.

4.2.1 Boundaries

Boundary regions are defined on cell faces by the user in STAR-CD. Any faces on the surface of the mesh which are not explicitly defined will be treated as a wall boundary. In many CFD codes including STAR-CD a special set of equations called wall functions, which are applied to wall boundaries in turbulent flows. These equations are used because near-wall flows for viscous fluids have different behaviour from the bulk flow, specifically, the fluid is stationary at the wall, called a 'no-slip' condition. As a result, a high shear 'boundary layer' exists near the walls where the fluid velocity changes rapidly from the wall to the bulk flow or free-stream.

Since velocities change greatly over this short distance from the wall, a very fine mesh would be required next to the wall in order to capture the flow details, as shown in Figure 4.1.



a) Fine mesh required without wall-functions

b) Coarser mesh may be used with wall-functions

Figure 4.1 Meshes required for calculating near-wall flows

The wall-function numerically approximates the flow changes through the boundary layer, however, and so the mesh does not have to be finer near the walls. This significantly reduces the number of cells needed to mesh a complex geometry which in turn reduces the amount of resources required to perform the analysis. Hence CFD becomes a more viable tool for a number of industrial applications.

STAR-CD uses a wall-function [58] defined for any cell that lies next to a wall as follows;

$$u^+ = \begin{cases} y^+ & , \quad y^+ \le y_m^+ \\ \frac{1}{\kappa} ln\left(E y^+\right) & , \quad y^+ > y_m^+ \end{cases} \tag{4.1}$$

where u^+ – $(u - u_w) / u_\tau$
u – tangential fluid velocity
u_w – wall velocity
u_τ – $(\tau_w / \rho)^{1/2}$
τ_w – wall shear stress
y^+ – $\rho C_\mu^{1/4} k^{1/2} y / \mu$
κ, E – empirical coefficients

One obvious disadvantage to using wall-functions is that they are an approximation to the near-wall flow. The boundary layer is not identical for all flows, nor does it even remain uniform throughout one flow domain. In reality, the boundary layer changes in its size, its flow regime, whether laminar or turbulent, and its composition of differing flow patterns such as shear layers. If the user accepts the approximations of the wall-functions, care must still be taken with the mesh size at the walls. Wall-functions in finite volume codes are applied only to those cells which lie on the walls of the mesh. This means that these cells must contain the entire boundary layer approximation. If the wall cells are too large, then the logarithmic velocity profile will be applied to too great a region of the flow field. Also, if the cells are too small, then the actual boundary layer may lie partially outside the cells with the wall-function approximations. Ideally; the wall cells will contain the entire boundary layer, but this is often difficult to achieve because the boundary layer thickness changes according to the geometry of the flow domain and sometimes changes in time too.

4.2.2 Solution algorithms

To solve the discretised flow equations throughout the mesh, STAR-CD offers two solution algorithms based upon the PISO [59] and the SIMPLE [33] algorithms. Both solution algorithms were developed for structured, hexahedral meshes with no cell distortion. STAR-CD uses modified versions of these algorithms such that unstructured meshes and some cell distortion can be accommodated. Generally, the STAR-CD version of the SIMPLE algorithm can accept more cell distortion than the modified PISO algorithm.

The PISO algorithm includes the time derivative in its calculations, and so can be used to solve both steady and transient flows, whereas the SIMPLE algorithm, as implemented by STAR-CD, deletes the time-varying derivatives from its calculations and so is only used for solving steady flows. Both solution algorithms use a predictor-corrector strategy for solving the discretised flow equations, where the flow equations are temporarily decoupled from each other so that they can be solved sequentially. In this technique, an initial field of variables is assumed, the predictor stage, and then the decoupled flow equations are solved giving new values of the flow variables. These new values are then used to modify the predictor values in the corrector stage.

The PISO algorithm as formulated for STAR-CD has more than one corrector stage, and these calculations are repeated until an internal measure of minimum error has been satisfied [58]. For steady flows the PISO algorithm uses a large time-step, δt, in order to accelerate convergence, but if this time-step is too large then numerical instabilities may occur in the calculation which prevent a solution from being reached. In order to address the problem of numerical instabilities, a method of 'under-relaxation' may be used. For this, the value of a variable, α_{new}, is modified to be the weighted mean of the previous value, α_{old}, and the newly predicted one, α_{calc}, where the under-relaxation factor, ω is the weighting factor for this calculation, as shown in Equation 4.2.

$$\alpha_{new} = \omega\alpha_{calc} + (1-\omega)\alpha_{old} \tag{4.2}$$

The SIMPLE algorithm is similar to the PISO algorithm, except that the former uses only one corrector stage. Again under-relaxation is employed to reduce numerical instabilities in the calculations. The computational model therefore includes an under-relaxation factor for each variable being solved for.

4.2.3 Convergence criteria

The flow calculations may converge to a solution, diverge or even oscillate between two stable conditions. The user can judge whether the analysis is converging by monitoring the 'residuals' that are reported for each iteration during calculation.

In STAR-CD, the momentum equations are linearised in the predictor stage to the form;

$$Ax = b \tag{4.3}$$

where the vector x is solved for using a matrix of operators A and a vector of known values, b. These equations are then solved iteratively in the corrector stage to produce a new value of the vector variables, x. Since the flow equations are non-linear, A is a function of x. Thus the updated values of x will produce a new matrix of operators A' and the residual error, r, is given by

$$r = b - A'x \tag{4.4}$$

Equations of the form of (4.3) are then again solved iteratively with the new values until r becomes smaller than a preset limit. For transient flows, STAR-CD uses a minimum internal measure of error but for steady flows the user must define the required residual limit. If the calculation converges, the residual error becomes smaller but if it diverges the residual errors will continue to grow. By monitoring the residuals reported for each iteration, the user can determine whether the calculations are converging to a solution or not. Often a diverging flow can be detected within ten iterations using STAR-CD and the analysis can be stopped and corrected without wasting a long run time.

As a further aid to judging convergence, STAR prints out the values of the flow variables at a monitoring location within the mesh at each iteration. The monitoring location is a cell selected by the user, and ideally this cell lies in a region of rapidly changing flow. When the calculations are near to convergence, the values of the flow variables will be changing very slowly, if at all, from one iteration to the next.

4.2.4 Cell distortion

There are a number of cell types supported by STAR-CD but the basic cell for the finite volume formulation is an hexahedral cell. Figure 4.2 shows a hexahedron with its eight vertices labelled. It can be seen from this Figure that vertices may be collapsed onto one another to produce other types of cells, such as a pyramidal pentahedron, a triangular prism and a tetrahedron.

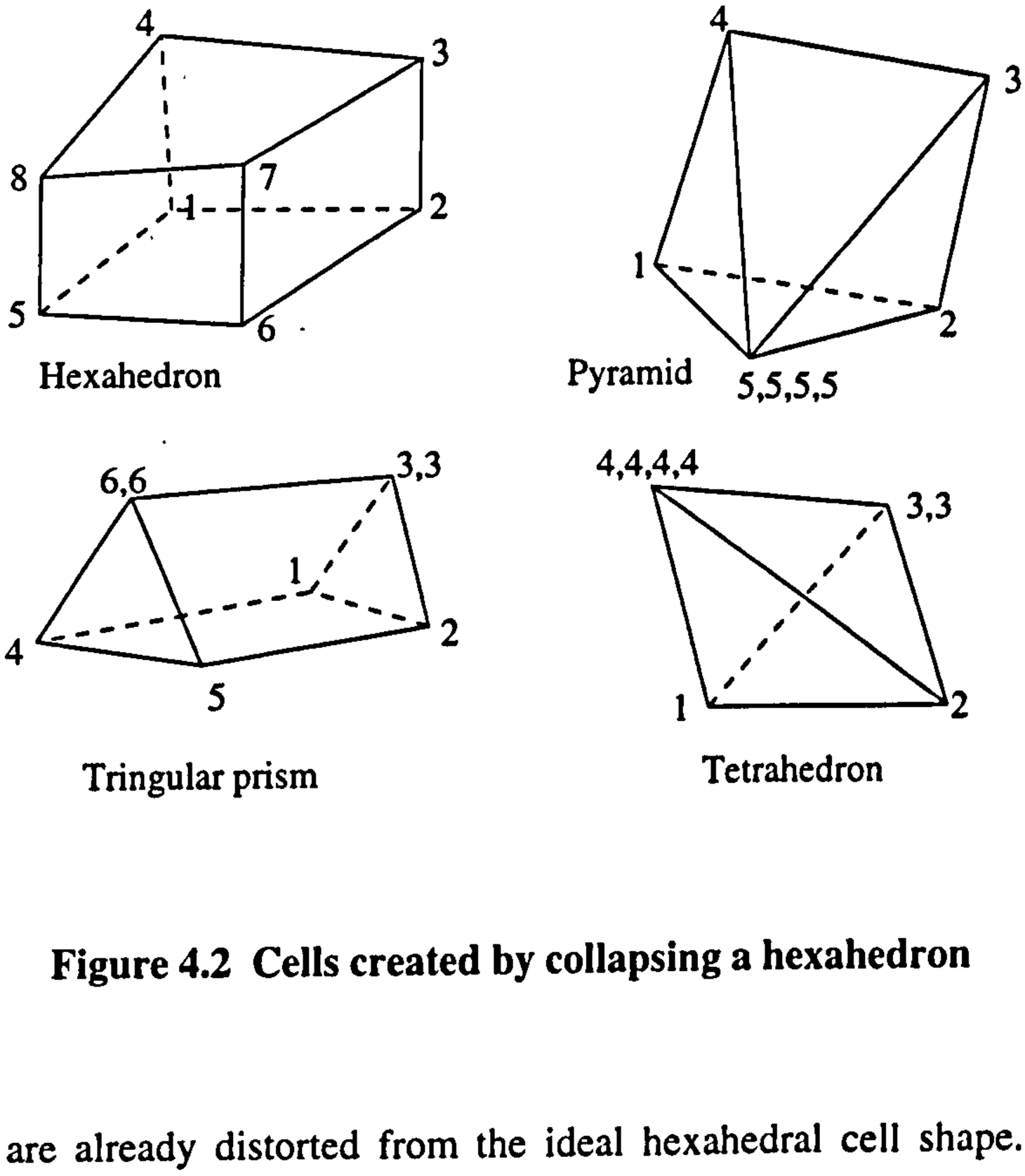


Figure 4.2 Cells created by collapsing a hexahedron

Collapsed cells are already distorted from the ideal hexahedral cell shape. It is therefore more likely that such cells will be of poorer quality than the neighbouring hexahedral cells. Consequently, care must be taken to ensure that collapsed cells do not lie in regions of rapid

variations in the flow variables, where problems of poor cell quality can be significant. Also the positioning of collapsed faces, that is, triangles, on the surface of the mesh must be avoided if at all possible, since the wall-functions are applied to these cells based upon quadrilateral faces. In view of these possible problems with the positioning and quality of collapsed cells, only hexahedral cells have been used in this work.

The computational models for the various meshes generated should be defined with consideration to STAR-CD's capabilities as outlined in this Section.

4.3 Creating the computational model for steady flows

The flow conditions to be added to the computational model were for the engine running at 1000 rpm. Experimental benchmarks for the CFD analysis were performed at Imperial College using a blowing rig which had air flowing through the port, past the valve at both 4mm and 8mm valve lifts, and out through the bottom of the cylinder. The piston is not included in this setup. As the air pressure is kept constant at the inlet and at the outlet, it is assumed that there is a steady flow through the port and cylinder geometry. The boundaries of the engine geometry were therefore defined in Table 4.1 below.

Boundary	Boundary condition	Where applied
Inlet	Constant velocity of 19.7 m/s	Port intake, velocity vector normal to the plane of the intake
Outlet	Constant static pressure of 1.0E5 Pa	Bottom of cylinder
Symmetry	Zero accelaeration and mass flow across boundary	Plane of symmetry

Table 4.1 Boundary regions applied to engine geometry

One velocity is set as an initial condition for the whole flow field. Since the in-cylinder flow is so complicated, any single initial flow condition will be arbitrary since it will be considerably different from the solution. Nevertheless, the port flow upstream of the valve has a definite bulk flow velocity and so the initial condition for the whole flow field was set to;

u = -10 m/s

v = 0.1 m/s

w = 0.1 m/s

where u,v,w respectively are vectors along the x, y and z co-ordinate axes of the global co-ordinate system. This gives the initial flow a bulk motion along the port and non-zero velocity in the other coordinate directions.

Other flow parameters defined in the computational model include which variables are solved. Velocity and pressure are always solved for, but others may be specified as constant throughout the domain if they are not to be solved. Thus if viscosity is not to be included, then a constant 'effective turbulent viscosity' is defined by the user, or a laminar viscosity defined if the flow is laminar. If viscosity is to be solved for, then the type of turbulence model needs to be specified. The 'k - ε' turbulence model used is generally based upon an initial value of the turbulence intensity, I, of 0.01

$$I=u/U=0.01 \tag{4.5}$$

where u = turbulent component of velocity, U = inlet velocity

This initial assumption that the turbulent component of velocity is 10% of the free stream velocity is common for in-cylinder flow fields [58]. Values for k and ε are then calculated according to the k - ε model as follows

$$k=u^2/2 \tag{4.6}$$

$$\varepsilon=(C_\mu^{3/4}k^{3/2})/l \tag{4.7}$$

where

C_μ = empirical coefficient (usually 0.009)

l = characteristic length

Using a characteristic length of 2.5E-3 m, which is approximately one tenth of the diameter of the port, equations 4.5 to 4.7 gave ;

$k = 1.08(m/s)^2$

$\varepsilon = 74.62\ J/m^3 s$.

Density can also be specified as a constant or it can be solved for using isobaric or ideal gas equations [58] and the latter is given in Equation 4.8

$$\rho = p/(RT(m_i/M_i)) \tag{4.8}$$

where

ρ = density

p = pressure

R = Universal gas constant

T = temperature

m_i = mass fraction of a constituent with molecular weight M_i

Other numerical parameters required in the computational model include;

i) residual-limits: usually 1.0E-4, as recommended in the STAR-CD manuals [58].

ii) under-relaxation factors (see equation 4.2):

velocity: 0.4

pressure: 0.05

k and ε: 0.03

viscosity: 0.95

density: 0.95

These are parameters to aid convergence and reflect the sensitivity of each variable. Hence

viscosity and density are fairly stable and are given large under-relaxation factors, whereas k and ε are generally more difficult to converge and so are given small under-relaxation factors.

iii) number of iterations to run: 300 initially, but a solution run could be restarted from a previous calculation.

4.4 Steady flow engine simulations

Steady flow runs were performed for mesh-2 and mesh-3. The results from these analyses are presented in this Section. Guided by the comparisons to experimental data, a new mesh, mesh-4 was created by extending the length of the cylinder in mesh-3 and by modifying the valve structure and position. Further analysis runs were then made and the results are also described in this Section, but first the general approach used for obtaining these solutions is outlined below.

4.4.1 Running the steady flow simulations

The initial conditions in the computational model differed greatly from the actual flow field since the latter were not known before the analysis began. Due to this large difference and to the complexity of the flow being simulated, the solver could not calculate the solution when velocity, turbulent viscosity and density are solved together from these initial conditions. Ideally, better initial conditions should be specified but it is also possible to use another approach in order to obtain a solution.

Potential flow calculations, that is, for a fluid flow having constant density, zero viscosity and zero vorticity, can be used to determine an initial flow field. In such conditions, no-slip conditions are not applied to the walls and flow separation will not occur. Indeed this method for obtaining initial flow conditions is used in other commercial CFD codes such as PHEONICS. Unfortunately, STAR-CD v2.1 does not have this capability and so yet another approach must be attempted.

First analysis runs were made solving only for velocity and pressure. A constant value for the density of air at room temperature and pressure, 0.981 kg/m^3 was used, along with a constant effective turbulent viscosity of 2.56E-3 m^2/s which is recommended for engine flows by Computational Dynamics [60]. These flow conditions were run for about 300 iterations after which, the flow field had improved and the velocity variables could be seen to be stabilising. The solution was judged to be stabilising when the residuals had dropped down from an initial value of around 1.0E3 by four orders of magnitude and when the variables at the monitoring location were changing very slowly, say on every third iteration when measured to four significant figures.

The constant effective viscosity term was then replaced with solving for viscosity using the k - ε turbulence model. The analysis run was then resumed from the previous solution obtained. Restarting the run, however introduces perturbations in the values of the residuals, and so the run must be continued until the residuals became substantially lower than their initial values once more.

At this point, the constant density term was replaced by the density solver using an ideal gas model. Since a cold flow was being analysed, the temperature, T, was set to a constant 293 K for its inclusion in the ideal gas equation. This completed the set of flow variables to be solved for, and the code was then run until convergence was judged to have occurred, that is, when the residuals were less than 1.0E-4, again as recommended by Computational Dynamics [58]. Thus the final results obtained were for steady, turbulent, compressible, cold flow though the port and cylinder at a range of valve lifts.

Using the above approaches and parameters, the various meshes generated were run to determine which would give the best results. Typical run times on Jaguar's CONVEX mainframe computer were 120 cpu hours, that is, dedicated computer time. Since the CONVEX is heavily used at Jaguar, analysis jobs share the computing time and so the time from submitting the job to obtaining the results can easily run to one week.

4.4.2 Running mesh-2

A computational model was created for mesh-2, that is, the hexahedral mesh generated from a tetrahedral mesh described in Section 3.4. The analysis run for this mesh proved unsuccessful, however, and the solution diverged after five iterations. When the results for the fourth iteration were viewed, extremely high velocities were shown for just a few cells which turned out to be the most distorted cells in the mesh.

A check of the original tetrahedral mesh showed some tetrahedra had internal angles of 179.3^o and so hexahedra created from these tetrahedra would be even more distorted. Thus a clean-up of the tetrahedral cells was attempted before the mesh was divided into hexahedra in the following way.

Some excessively distorted tetrahedra on the inlet boundary could be deleted without any significant change to the geometry of the mesh at the inlet itself. Other distorted tetrahedra were improved by moving vertex positions to give better shaped cells. A new mesh of hexahedra was created from this modified tetrahedral mesh, but this mesh also failed to run in STAR-CD. Once again excessive velocities were calculated for the most distorted cells in the mesh, which had internal angles of 178.8^o. The tetrahedral mesh generated using I-DEAS V was concluded to be unsuitable for this approach since the resulting hexahedral mesh needed too much cleaning-up before it could be run using STAR-CD. That is, whilst the method of generating such a mesh is initially very quick and easy, the mesh simply cannot be used by STAR-CD

Interestingly, mesh-2 was successfully run using another CFD code called FIDAP [61]. This code is based upon the 'finite element' formulation and the latter is described by Reddy [62]. The drawback of this commercial finite element code is that it takes significantly more time and computing resources to run than STAR-CD for the same mesh. This is the main reason why such commercial finite element codes were not selected for this engine work, despite such codes being more tolerant of cell distortion as was demonstrated.

4.4.3 Running mesh-3

Two valve lifts for mesh-3 were run as steady flow cases, at the 6mm and 7.5mm valve lifts, since these were the first meshes to be created from model-3. When comparing the results to experiment, however, it was found that the valve lifts had been measured in different ways in the computational model and experiment which meant that mesh-3 actually had valve lifts 1.2mm smaller than the corresponding experiments. Thus the previously taken CFD valve lifts of 6mm and 7.5mm were actually 4.8mm and 6.3mm respectively.

The results obtained from the 6.3mm valve lift case were compared to the experimental results for an 8mm valve lift. This is obviously a large difference in the valve lifts, 1.7mm in fact, but Figure 4.3 shows that the swirl number changes on a steady curve between these valve lifts without switching from positive to negative values.

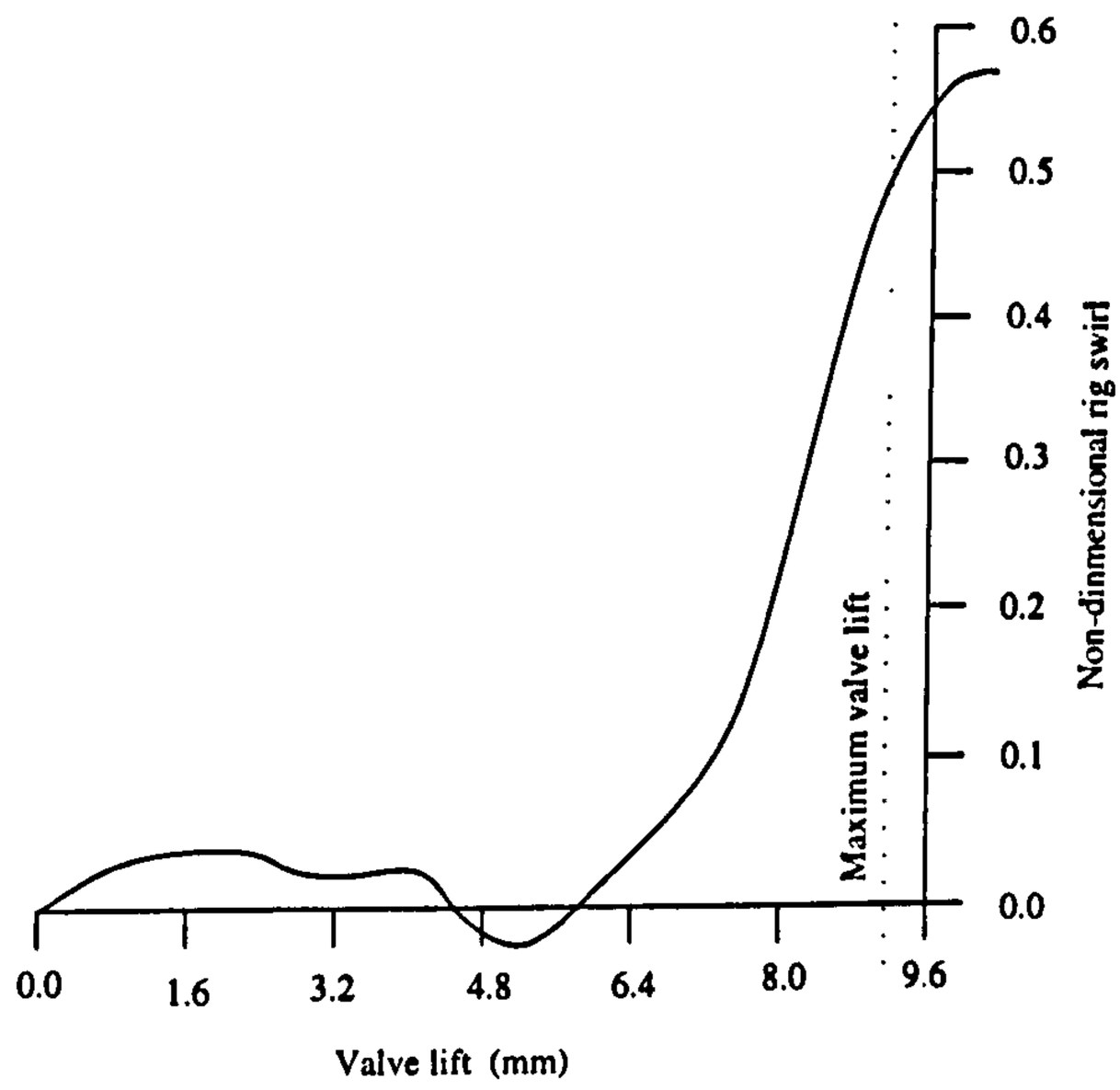


Figure 4.3 Swirl number for mesh-3

A change of sign for the swirl number is symptomatic of a change in the flow structure [26]. Since the sign of the swirl number does not change between the 6.3mm and 8mm valve lifts, it is assumed that the general in-cylinder flow structure is similar for both valve lifts. In view of this, the general trends between the two sets of results can be compared.

This comparison showed a fair correlation between the flow fields at the top of the cylinder, but at the bottom of the cylinder the results were distinctly different and a vortex is seen near the bottom of the cylinder in the CFD predictions. The same vortex structure was also seen for the 4.8mm valve lift predictions which is shown in Figure 4.4 but no such feature was found in the experiments.

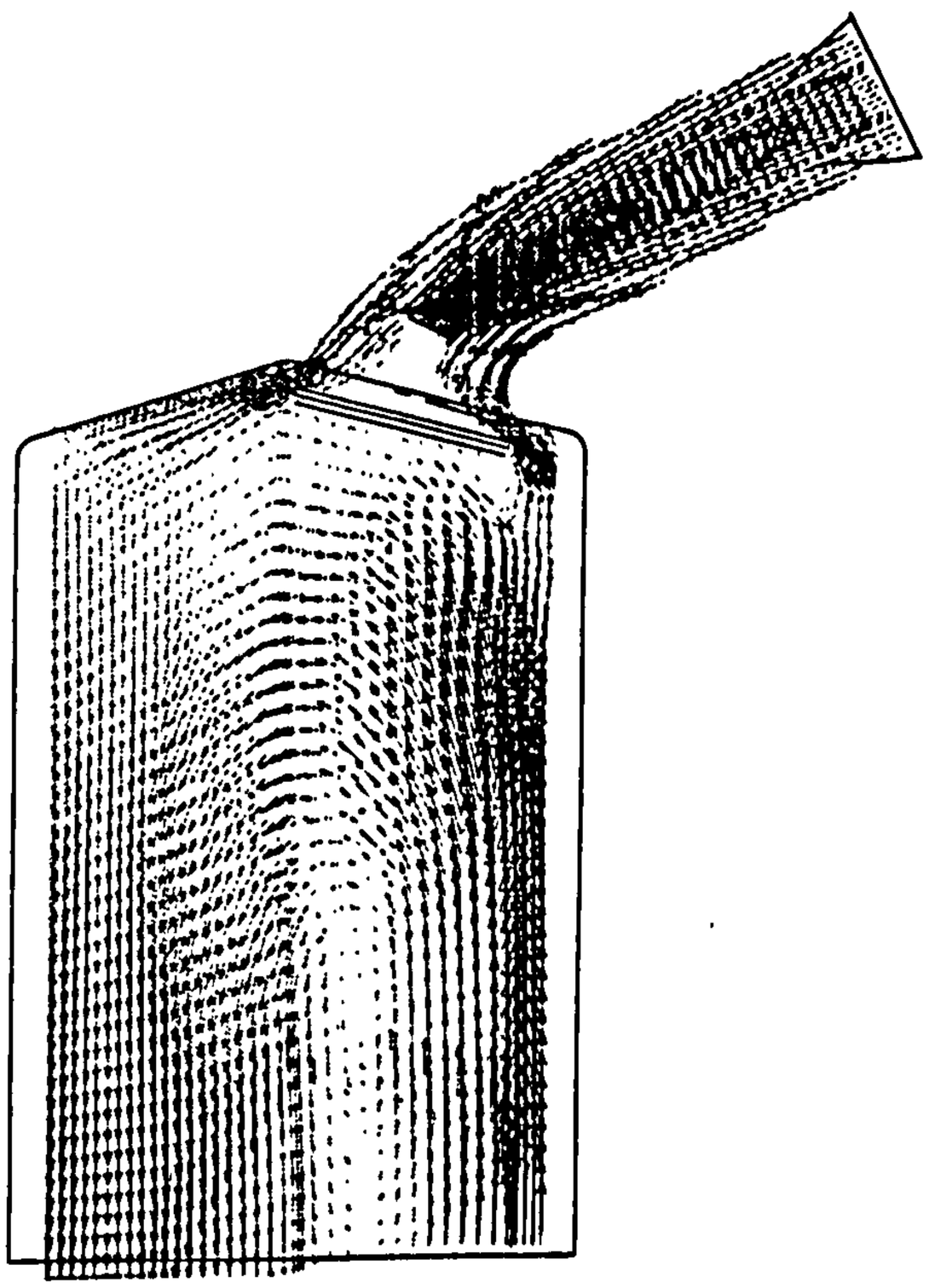


Figure 4.4 Vortex in cylinder at 4.8mm valve lift for mesh-3

These differences in flow structure can be attributed to two factors, the length of the cylinders and the boundary conditions for the CFD run. Firstly, the engine stroke was 0.12m and so mesh-3 was created with a cylinder 0.12m in length. In the experimental rig, however, the

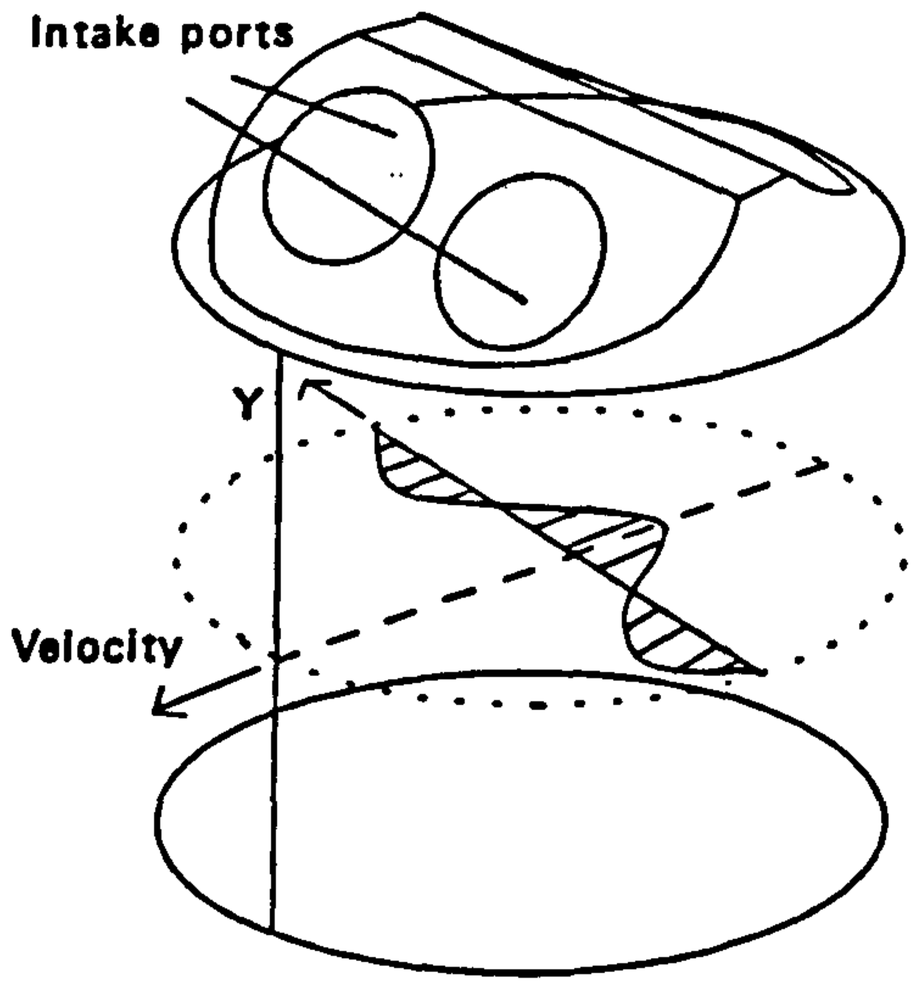
cylinder was extended by 0.4m to ensure that the steady in-cylinder flow measured was not influenced by end-effects of the cylinder. This meant that there was a difference between the experimental and computational geometries, and so no detailed comparison of these results would be valid.

The second factor was a pressure boundary condition that had been defined at the bottom of the cylinder in the CFD simulation. A pressure boundary imposes constant static pressure on the flow across the boundary region, and so an accelerating flow would not be properly modelled here. Indeed, this artificial pressure constraint created the strong vortex predicted at the bottom of the cylinder which were not seen in the experiments.

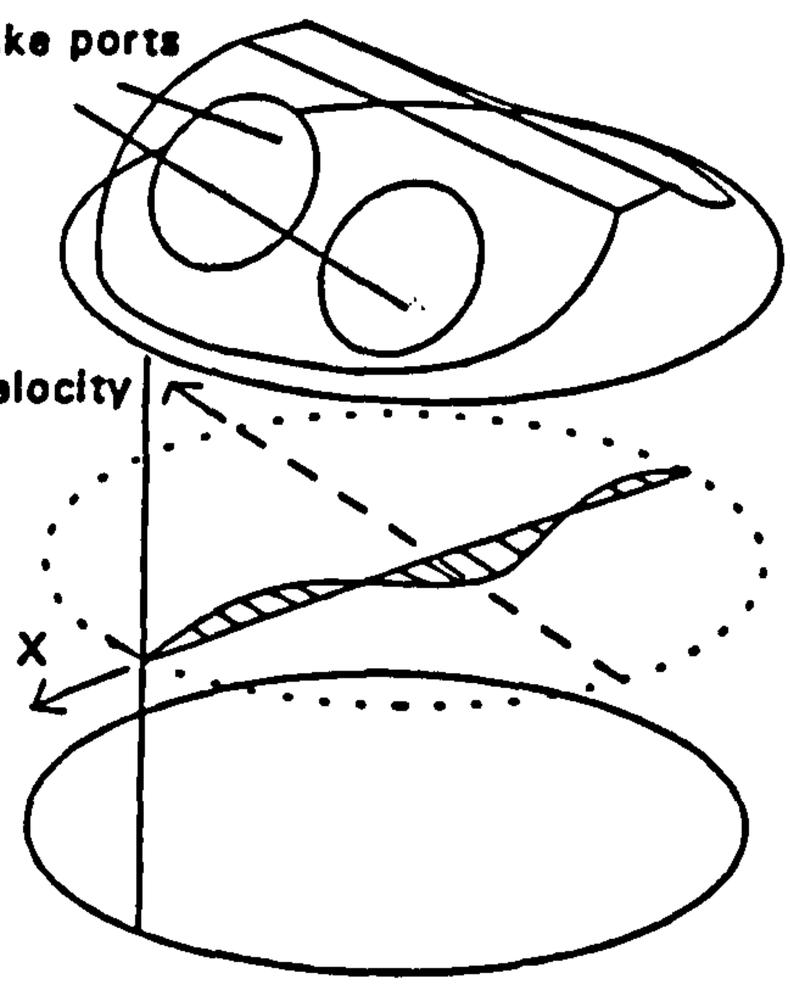
4.4.4 Creating and running mesh-4

On the basis of the results described above, a new mesh, mesh-4, was created. The methods used to create mesh-3, as described in Section 3.5, were repeated to obtain mesh-4, which had an extended cylinder and valve lifts matching the experimental rig. The near-valve area was also redesigned slightly to allow three valve lifts of 4mm, 6mm and 8mm using the automatic vertex-moving program. The meshes thereby followed the experimental geometry closely. These meshes were then run using STAR-CD with the flow parameters and techniques described earlier in Section 4.4.1.

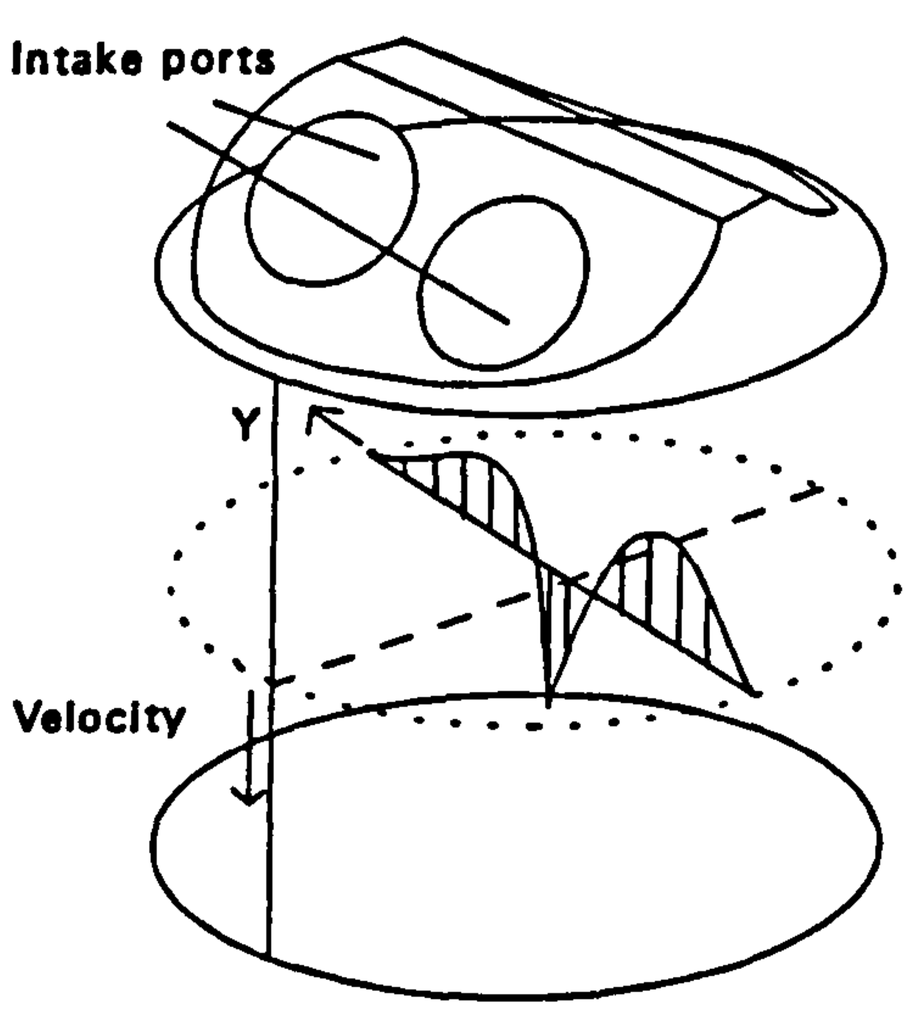
In order to compare to experiment, CFD predictions of velocity were obtained by interpolating the velocity data in the mesh cells within which the measurement points lie. These LDV measurements were made in three planes parallel to the piston head at 7.5mm, 40mm and 85mm below the position of the piston at TDC. CFD and LDV results are compared by isolating the components of the flow velocity along these measuring planes in a number of directions which are defined in Figure 4.5. In this Figure, the measuring planes are parallel to the x-y coordinate plane and the cylinder's symmetry plane is the x-z coordinate plane.



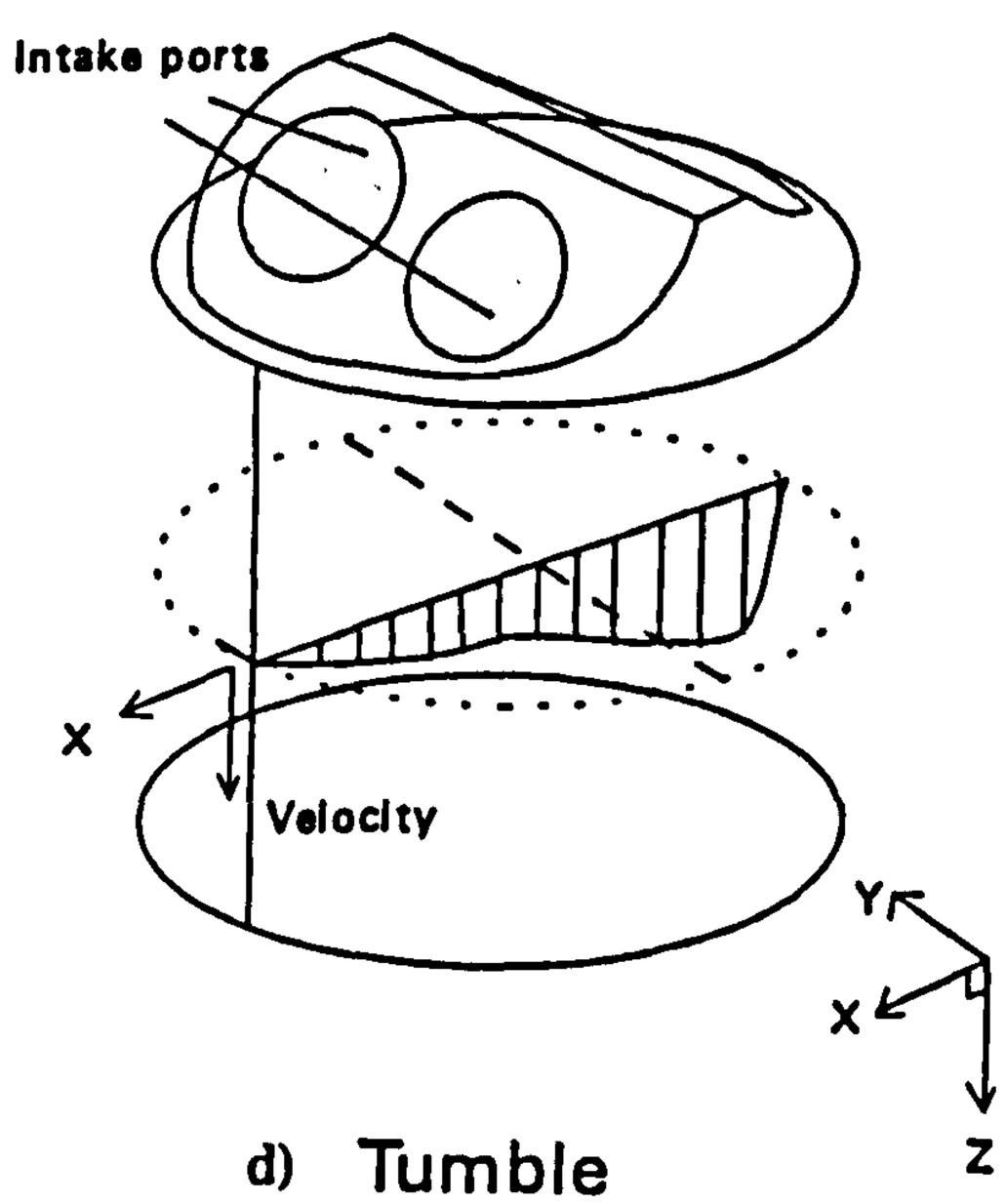
a) Swirl x-direction



b) Swirl y-direction



c) Cross-tumble



d) Tumble

Figure 4.5 Directions for flow measurements

In-cylinder flow in the x-y plane is referred to as swirl. In this work the swirl is further broken down into components in the y-direction, normal to the symmetry plane, and the x-direction, parallel to the symmetry plane, as shown in Figures 4.5a and 4.5b respectively. The flow in the y-z plane is referred to here as the cross-tumble and is illustrated in Figure 4.5c. This flow direction is normal to both the measuring plane and the symmetry plane. The flow in the symmetry plane itself is called the tumble and is illustrated in Figure 4.5d.

Since there are three measuring planes and four components of velocity at each plane, then for each valve lift there are twelve sets of velocity comparisons. Figure 4.6 over-leaf shows a graphical comparison between experimental and CFD results for each of the four components of velocity at each measuring plane. Note that the rig measurements were taken across the entire diameter of the cylinder, whereas the CFD results are only available for one half of the cylinder since a symmetric flow field was assumed. In order for a complete comparison to be made between CFD and LDV results, the CFD results are reflected about the symmetry plane for cross-tumble and for swirl in the x-direction. These comparisons are presented in the following subsections for the 4mm and 8mm valve lift cases respectively.

4.4.4.1 4mm valve lift case

First considering the swirl in the y-direction, as shown in Figure 4.6a. At each of the measuring planes the CFD results give near-zero flow across the symmetry plane. This is because a boundary condition of zero-mass flow had been imposed across the symmetry plane based upon the assumption that the symmetric geometry would create a symmetric flow field within the cylinder, as described in Section 1.2.2. However the LDV measurements showed a significant flow across the symmetry plane at the 7.5mm measuring plane, particularly on the opposite side of the cylinder from the intake valves. This indicates that the assumption of a symmetric flow field was invalid for the 4mm valve lift case, at least near the valve. At the 40mm and 85mm measuring planes, a good correlation was found where both CFD and experimental results show near-zero flow across the symmetry plane.

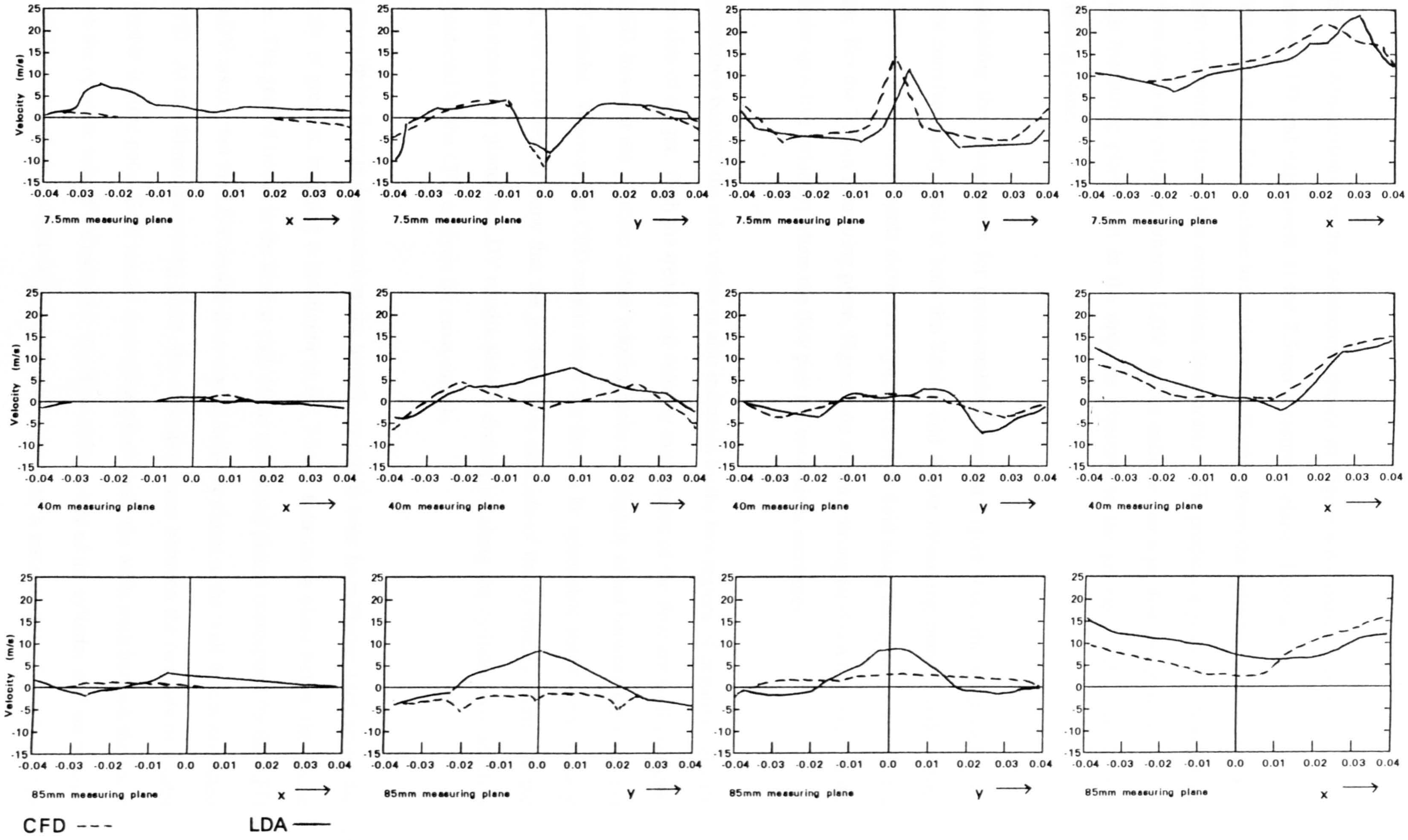


Figure 4.6 Results comparisons for 4mm valve lift case

The swirl comparisons for the x-direction shown in Figure 4.6b show a good correlation between CFD and experiment at the 7.5mm measuring plane. Here a strong jet exits the intake port and the flow is close to symmetrical. Further down the cylinder, at the 40mm and 85mm measuring planes, this correlation deteriorates. CFD predicts a gradually diminishing jet flow down the cylinder, whereas LDV results indicate that a jet flow exists at 40mm and 85mm measuring planes but in the opposite direction to the jet measured at the 7.5mm measuring plane.

Considering the comparisons for cross-tumble shown in Figure 4.6c, the CFD and LDV results correlate fairly well at both the 7.5mm and 40mm measuring planes. Indeed both predictions and measurements show near-symmetric flow field along the cylinder axis in this plane. For the 7.5mm measuring plane, Figure 4.6c shows a strong jet down the centre of the cylinder (positive velocity) where the flow past the two valves combines.

Recirculation beneath the inlet valves is also indicated by the two regions of negative velocity either side of the jet. Both the trends and velocity magnitudes of the flow are well predicted by CFD, however the positions of the velocity peaks are slightly offset between the CFD and LDV results. Whereas the CFD results show the flow to be symmetric about the symmetry plane, the LDV results show that the jet flow is to one side of the symmetry plane. At the 85mm measuring plane the LDV results show a distinct jet along the cylinder axis which is not predicted by the CFD analysis for cross-tumble.

The in-cylinder flow is dominated by the tumble, as can be seen from Figure 4.6d, where the velocity is greatest, reaching a maximum at the 7.5mm measuring plane below the intake valve. The general trends in the flow at each of the measuring planes correspond for the CFD and LDV results, but the LDV results show a peak velocity closer to the wall than is predicted by CFD. At the 40mm measuring plane, the correspondence between the two sets of results for tumble is quite good. Both results show strong flows near the walls both beneath the valve and on the opposite side of the cylinder. The flow at the centre of the cylinder is close to zero, and LDV results show a slight negative velocity indicating a recirculation of the flow. The

correlation at the 85mm measuring plane is that both sets of results show strong flow down the cylinder at the walls. However the CFD results predict a near wall velocity opposite the intake valves that is lower than the LDV measurements and CFD also predicts a near wall flow beneath the intake valves that is stronger than the LDV measurements. At the centre of the cylinder in this measuring plane, a difference of up to 7m/s can be seen between the CFD and LDV results.

In summary for the 4mm valve lift, the comparison of results have shown some good correlations and some poor correlations. No one measuring plane or flow direction showed consistently good or bad CFD results. Neither did regions of similar flow structure such as strong jets give consistently good or bad results. However, at velocities greater than 10m/s CFD has predicted the flow trends reasonably well.

4.4.4.2 8mm valve lift case

All velocity comparisons for the 8mm valve lift case can be seen in Figure 4.7 over-leaf. The flow across the symmetry plane, that is the swirl in the y-direction as shown in Figure 4.7a, was close to zero for both CFD and experiments, at each of the three measuring planes. Thus the assumption that the flow field is symmetric, as mentioned in Section 1.2.2, may be valid for the 8mm valve lift case.

The correlation between CFD and experiment was not close for swirl in the x-direction, Figure 4.7b. Here the CFD results predict a symmetric flow field whereas the LDV results showed some asymmetry in the flow parallel to the symmetry plane, particularly at the 40mm measuring plane. At the 7.5mm measuring plane, LDV results show a distinct jet between the intake valves towards the opposite wall of the cylinder. Whilst this jet is predicted by CFD, the predicted magnitude is significantly less than that measured. Both CFD and LDV results show the same trends of a distinct jet flow back towards the wall beneath the intake valves at the 85mm measuring plane. Quantitatively, the CFD results tend to 'flatten' this peak as the jet and the near-wall flow (negative velocity) is under-predicted.

Figure 4.7 Results comparisons for 8mm valve lift case

Marked differences between CFD and experiment can be seen in Figure 4.7c for the cross-tumble at the 7.5mm measuring plane, where the LDV results show a stronger jet flow down the cylinder axis than that predicted by CFD. However the general flow trends of a jet flow down the cylinder between the valves and recirculating regions beneath the valves are predicted by CFD. At the 40mm and 85mm measuring planes this jet is not as strong, and the CFD and LDV results match more closely.

At the 40mm measuring plane the correlation for tumble is quite good, although CFD under-predicts the strong near wall flow opposite the valves as can be seen in Figure 4.7d. By contrast, at the 7.5mm measuring plane the CFD and LDV results are very different. Whilst both results show a velocity maximum beneath the intake valves, the magnitude of this velocity is significantly under-predicted by CFD. At the centre of the cylinder the results disagree greatly but the magnitude of the near-wall flow velocity opposite the intake valves correlates fairly well. The general flow trends are predicted fairly well at the 85mm measuring plane but CFD gives a near constant under-prediction of the velocity of about 5m/s.

As for the 4mm valve lift case, there was no one measuring plane, flow direction or flow structure that was consistently good or bad. So it cannot be said, for example, that the CFD predictions are generally good but fail to predict a certain feature. Only one trend is noticeable in these comparisons in that where CFD does not correlate well with experiment, the CFD results tend to under-predict velocity magnitudes. This is a 'damping' effect where peaks in flow velocity are flattened.

The conclusion then is that a complex three dimensional flow structure exists in the cylinder that is not fully described in detail by measurements along two axes at three measuring planes. Indeed a better validation for general flow structures may be obtained using flow visualisation techniques. This could show whether the overall flow structures predicted in the cylinder are also seen experimentally. If this is the case, then details of where the CFD performs poorly can be identified for further study.

4.5 Discussion of Comparisons

4.5.1 Quality of results

An overall view of the comparison of results between the CFD and LDV experimental data showed some good correlation and some poor correlations. There are a number of reasons why the correlation is not better, however, and these factors are discussed here.

One obvious reason why the CFD results and the experimental results do not match exactly is the turbulence model used in the CFD code. STAR-CD uses the k - ε model of turbulence which contains empirical constants derived from experiments that do not necessarily apply to engine flows. In addition, the k - ε model is known to be unsuitable for highly swirling and separated flows as it tends to over-predict viscosity. This is because the model assumes a high Reynolds number turbulent flow, whereas near-wall flow separation is associated with a low Reynolds number. This is a possible reason for the general under-prediction of jet flow peak velocities seen in the comparisons.

Another factor in the modelling of the flow geometry is the coarseness of the mesh. Typical mesh sizes for engine flows such as those analysed here are about 200,000 cells. But the mesh used for this flow analysis had only around 100,000 cells, and so the results can be expected to be poorly predicted with regards to the quantitative results achieved.

A further consideration is the assumed symmetry of the flow, mentioned in Section 1.2.2. Experimental measurements shown in Figures 4.6a and 4.7a indicate that the flow was not symmetric for all measuring planes and valve lifts. Within an actual engine, variations between the two intake ports and valves which lie within acceptable manufacturing tolerances may be sufficient to produce a different mass flow through each intake port. Thus even modelling the entire 3D geometry may not resolve this question, since the mesh will be identical each side of the symmetry plane. This problem may be partially avoided, however, if the mass flow through each intake port, after the bifurcation, was measured. These mass flow

rates could then be used as boundary conditions for analysis. Even so, differences in the valve mechanisms, such as cam profiles, can give rise to slightly different valve lifts and this change in geometry may also have a substantial effect upon the flow field.

4.5.2 Discussion of the flow field

Figures 4.8, 4.9 and 4.10 show the predicted flow structure down the cylinder for the 4mm, 6mm and 8mm valve lift meshes respectively. The 4mm valve lift case shows a series of vortices along the cylinder, whereas the 6mm and 8mm valve lift cases show no such vortices, only flow parallel to the cylinder axis. This suggests that for this port-valve-cylinder geometry it is mainly the energy injected into the flow at a small valve lift that creates swirling flow structures along the length of the cylinder, since very little swirling flow can be seen to be generated at a large valve lift.

These CFD results could also suggest that in the real engine a tumble motion may be created in the combustion chamber at small valve lifts but this tumbling motion may not be enhanced at higher valve lifts by the port, valve and roof geometry. Of course, this is a steady flow study at fixed valve lifts and the real transient flow can only be properly determined through a transient study of the geometry.

A dominant flow feature of all of these valve lift cases is the the strong air-flow directly down the cylinder walls opposite the intake valves, as can be seen in Figures 4.8 to 4.10, and in detail in Figure 4.11. The trends can prove valuable to engine designers, since, for example, a strong flow down the cylinder walls might cause fuel to condense on the walls and leak from the bottom of the cylinder. By examining the flow structure predicted by CFD, see Figure 4.11, it may be suggested that the angle of the port needs to be altered or that the shape of the valve should be changed. Indeed, a further simulation with these alterations to the geometry may well show that this flow feature has disappeared.

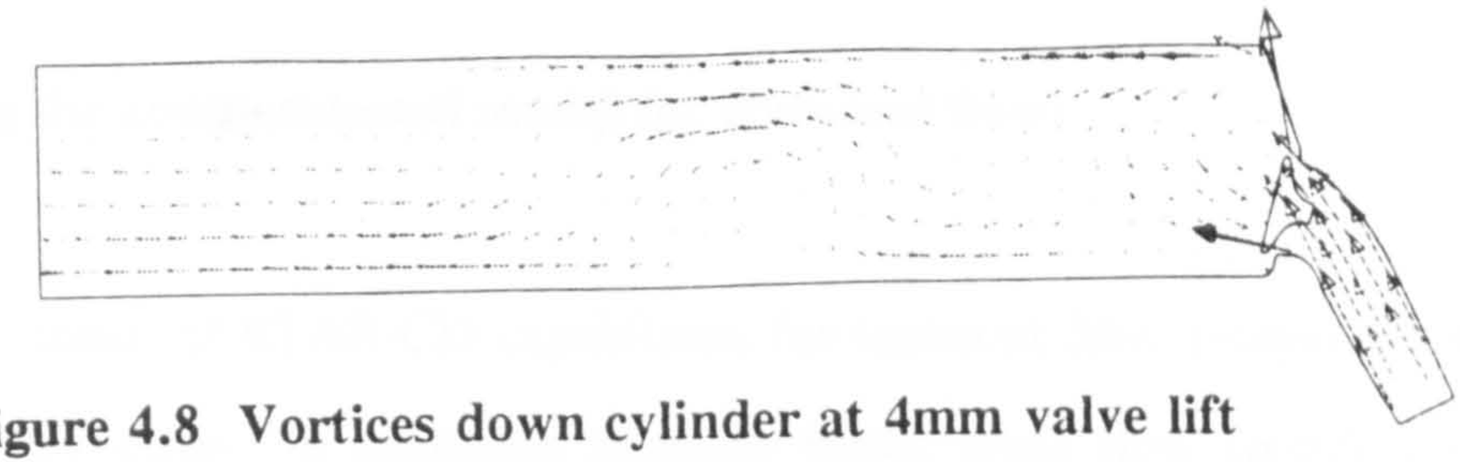


Figure 4.8 Vortices down cylinder at 4mm valve lift

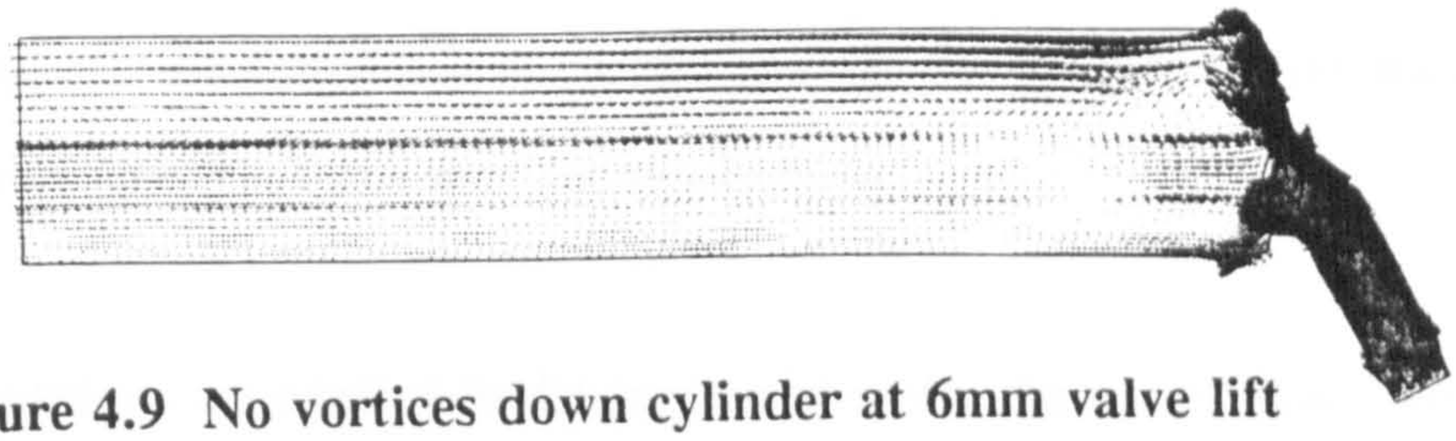


Figure 4.9 No vortices down cylinder at 6mm valve lift

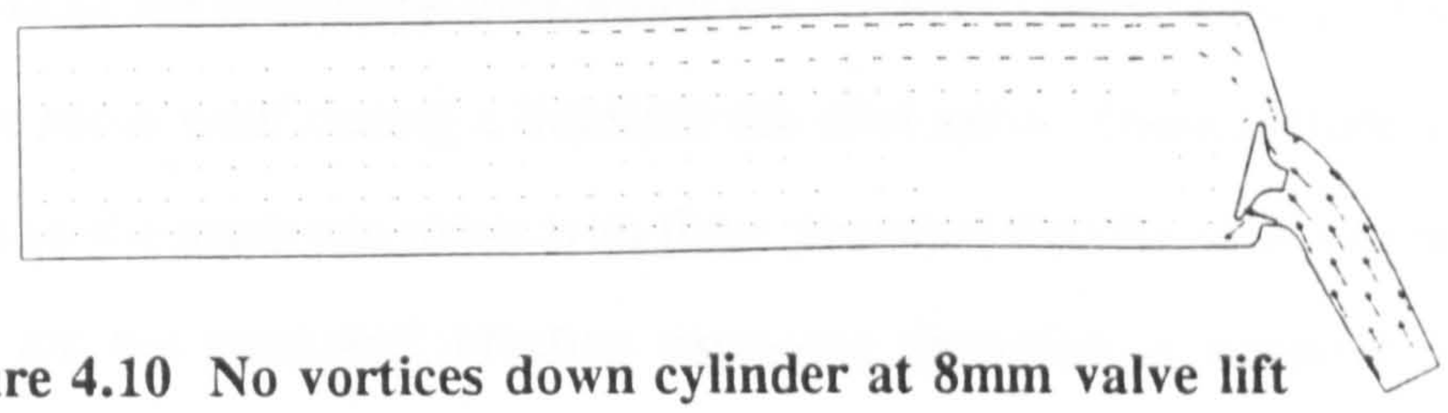


Figure 4.10 No vortices down cylinder at 8mm valve lift

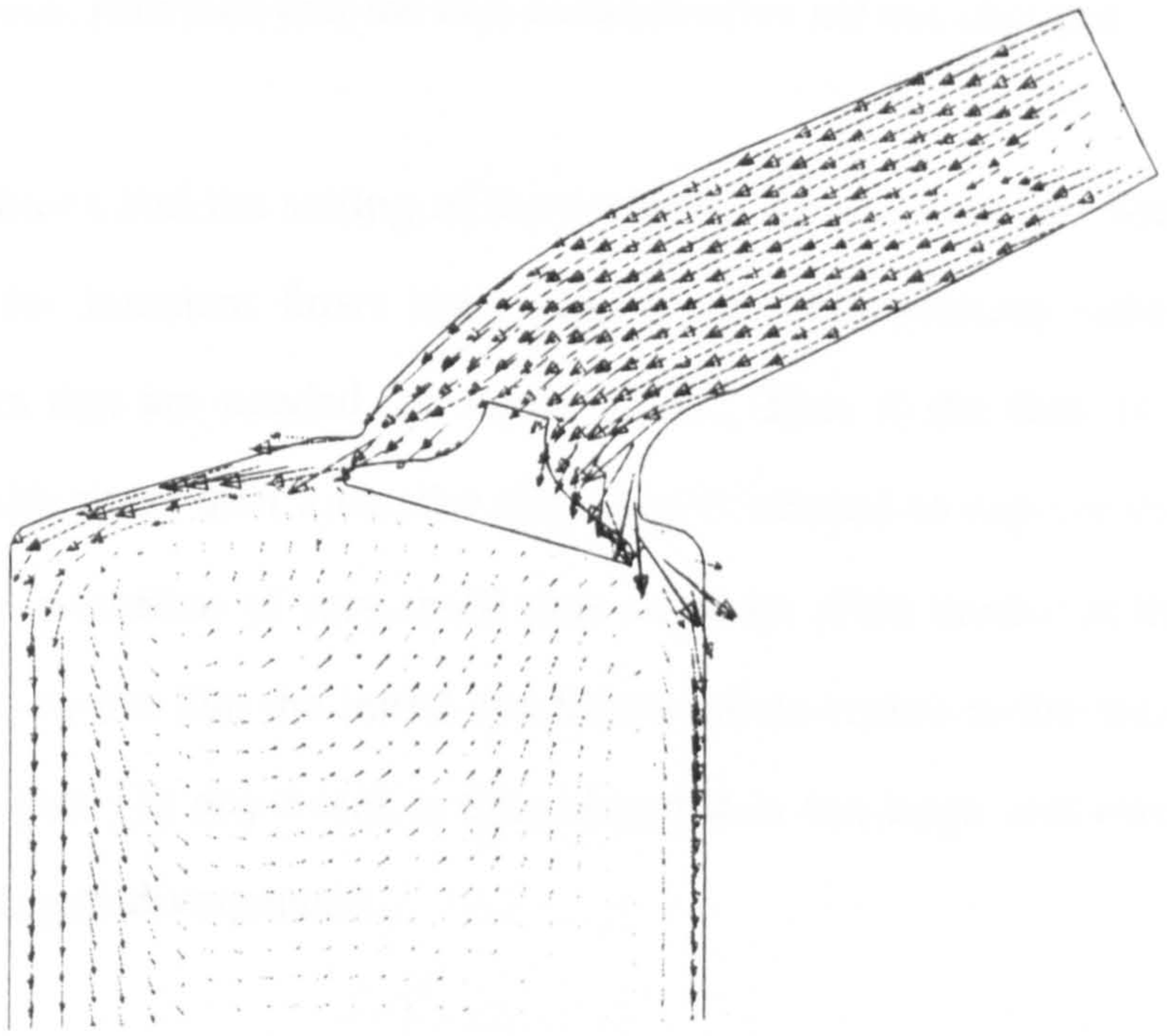


Figure 4.11 In-cylinder vortices near the valve

4.6 Creating the computational model for transient flows

A brief assessment of STAR-CD capabilities for transient flow analysis of engines was also included in this study. In analysing transient flows, some flow specifications are different from the steady flow case. Mesh generation is done in the same way as before, but the PISO algorithm is used to solve the discretised flow equations for transient flow and so, as mentioned previously in Section 4.2.2, any mesh generated for transient flow must be of a better quality than a comparable mesh used for steady flow.

Boundary conditions are applied in the same way as for the steady flow, but further options are available. New sets of boundary conditions can be imposed at any time during the transient flow analysis run through 'load-steps'. These define the boundary conditions and the number and size of the time-steps over which the boundary conditions are imposed. Options for altering the mesh itself during a transient run also exist. These include a moving mesh, where vertices in the mesh can move with time, provided that the cells which are defined by these vertices are not corrupted, creating excessive distortion or negative volumes. Other features for moving meshes include deletion or addition of layers of cells within the mesh, the deletion or addition of a whole section of the mesh and the complete redefinition of all the vertices in the mesh, provided that the cell connectivities are not changed.

The initial conditions and the setting of the variables are the same for both steady and transient flow, but for transient flows time-steps control the solution rather than the under-relaxation factors that are needed for steady flows. Thus if the flow is under-going some rapid changes with time, then a smaller time-step is needed to capture these changes effectively. Similarly, a number of very small time-steps are often needed at the beginning of the transient flow, to ensure that the initial conditions are corrected to the actual transient results in a small time-span. In any event, a time-step that is too large will often cause numerical instabilities leading to divergence.

4.7 Transient Flow Simulations

The solution of transient flow is straightforward, as has been described in Section 4.2.2, but the mesh of the port and cylinder during the engine cycle is much more complex than that used for steady flow analysis since it must continually change during the cycle to reflect the changing geometry. In this Section, methods for obtaining transient flow solutions for the moving piston and valve are discussed, along with some simple examples of the application of these techniques.

4.7.1 Piston movement

To simulate piston movement, the cells at the bottom of the cylinder must be moved up and down. In between each time-step in the transient solution, the coordinates of the vertices are altered so that they stretch the cells near the bottom of the mesh, effectively moving the lower boundary. The movement of the mesh can therefore be linked-in to the piston speed at various crank angles using the 'NEWXYZ' subroutine provided as user-coding in STAR-CD. This subroutine takes the current coordinates of all the vertices in the mesh and modifies them according to a mathematical description written into the subroutine by the user. NEWXYZ then returns these new coordinates to STAR for use in the flow calculations at that time-step.

There are, however, limits to the amount of distortion of the cells that is acceptable in STAR, in particular, the aspect ratio which is altered at each time-step for those cells being stretched or compressed. Thus for a fine mesh in the cylinder, which is usually required for engine flow studies, some cells must be deleted from or added to the mesh as the piston moves in order to avoid excessive aspect ratios.

The method for moving meshes is to list the cells to be added or deleted at which specific time-step in the transient solution in a user input file, 'file2'. Thus cells may be stretched for a few time steps using the NEWXYZ subroutine, and when further stretching would result in over-distorted cells, a new layer of cells can be added, and over the next few time-steps this

added layer is also stretched. The process continues until the maximum extension of the mesh is reached, and then the whole process is reversed, deleting and compressing cells using the same subroutines.

The current form of cell addition and deletion in the file27 is by operating on layers of cells and so a block structured mesh must be used in the region of mesh moving. This method is therefore only used easily for modelling the piston movement where the cylinder geometry is simple. Modelling the valve movement is a much more complex process and other methods must be used.

4.7.2 Valve movement

Two methods are presented here to achieve the full valve movement in an unstructured mesh during a transient flow case. Firstly, the automatic vertex moving algorithm used to generate the 4mm and 8mm valve lift steady flow cases, as described in Section 3.7.2, could be written into the NEWXYZ subroutine in ufile.f. But this could distort some cells too much, as was seen when running the steady flow cases. The other alternative, then, is to create new meshes for a series of valve lifts. This would again be achieved by using the automatic vertex-moving algorithm, but the resulting meshes could be checked first and possibly modified to ensure acceptable cell distortion. A new mesh can be input into the transient solution at any time during its running using file27, provided the cell connectivities remain the same. That is, the new mesh must have the same number of vertices and cells, and the same cell definitions as the old mesh it replaces.

Valve closure cannot be achieved simply by moving the vertices closer together near the valve seat, or by deleting the cells in that region. This is because the valve closure/opening is a sudden change in the geometry of the flow field being considered and not one which can be approximated by compressing cells. In STAR-CD a specific method is used, called 'block deletion' and 'block addition', which disconnects or reconnects two regions of flow. Thus the block deleted cells will simply be removed from the calculation, and entered in to the

calculations again with block addition. Very small time steps must be used around the time when a block deletion or block addition is performed, to ensure that the very rapid changes in the flow field to not destabilise the solution.

4.7.3 Two-dimensional transient simulations

A simple 2D model was created to illustrate the cell stretching and compressing, and the cell addition and deletion capabilities of STAR-CD. The mesh is one cell thick, and so effectively 2D for STAR-CD's purposes, and all of the cells are the same size. A pressure boundary condition has been defined at the inlet, and a wall boundary condition assigned to the lower faces. The movement of the bottom wall downwards will draw in air through the inlet, very crudely simulating a descending piston. Figure 4.12 shows results for cells stretched by the NEWXYZ subroutine written for this simple case.

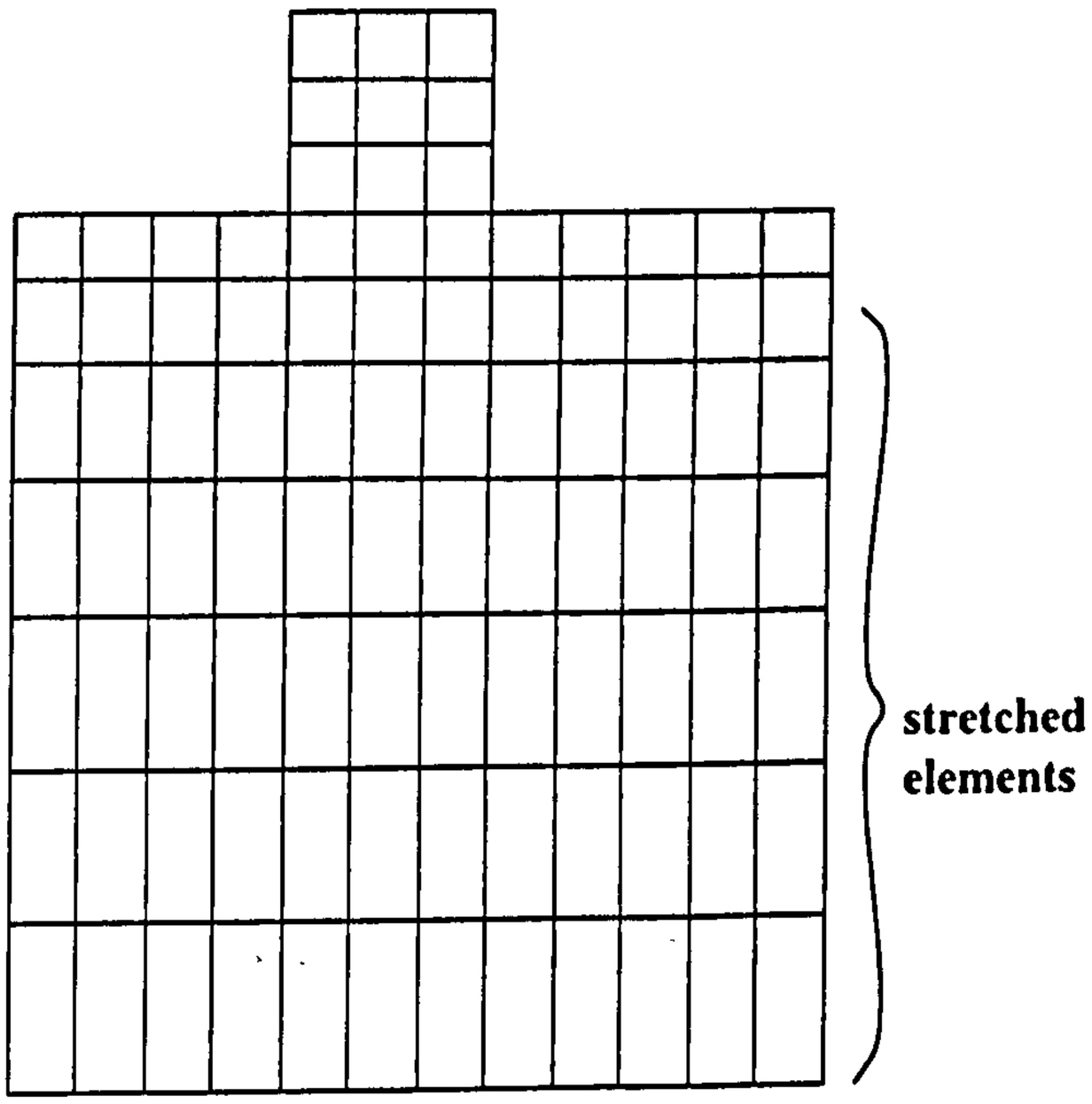


Figure 4.12 Cells stretched in 2D

The same 2D test case was then run again, but in this case cells were deleted as the mesh contracted, and added again when the mesh was stretched. This proved that the moving mesh capabilities in STAR-CD were suitable for simulating a moving piston in 2D.

4.7.4 Three-dimensional transient simulations

A coarse mesh, mesh-5, containing 2200 cells was tested for 3D transient flows to establish the mesh moving applications of STAR-CD. The mesh was first tested for viability by running it as a steady flow case. Mesh-5 was initially created with the boundary faces representing the piston head set at the piston's central position in the cylinder, as shown in Figure 4.13.

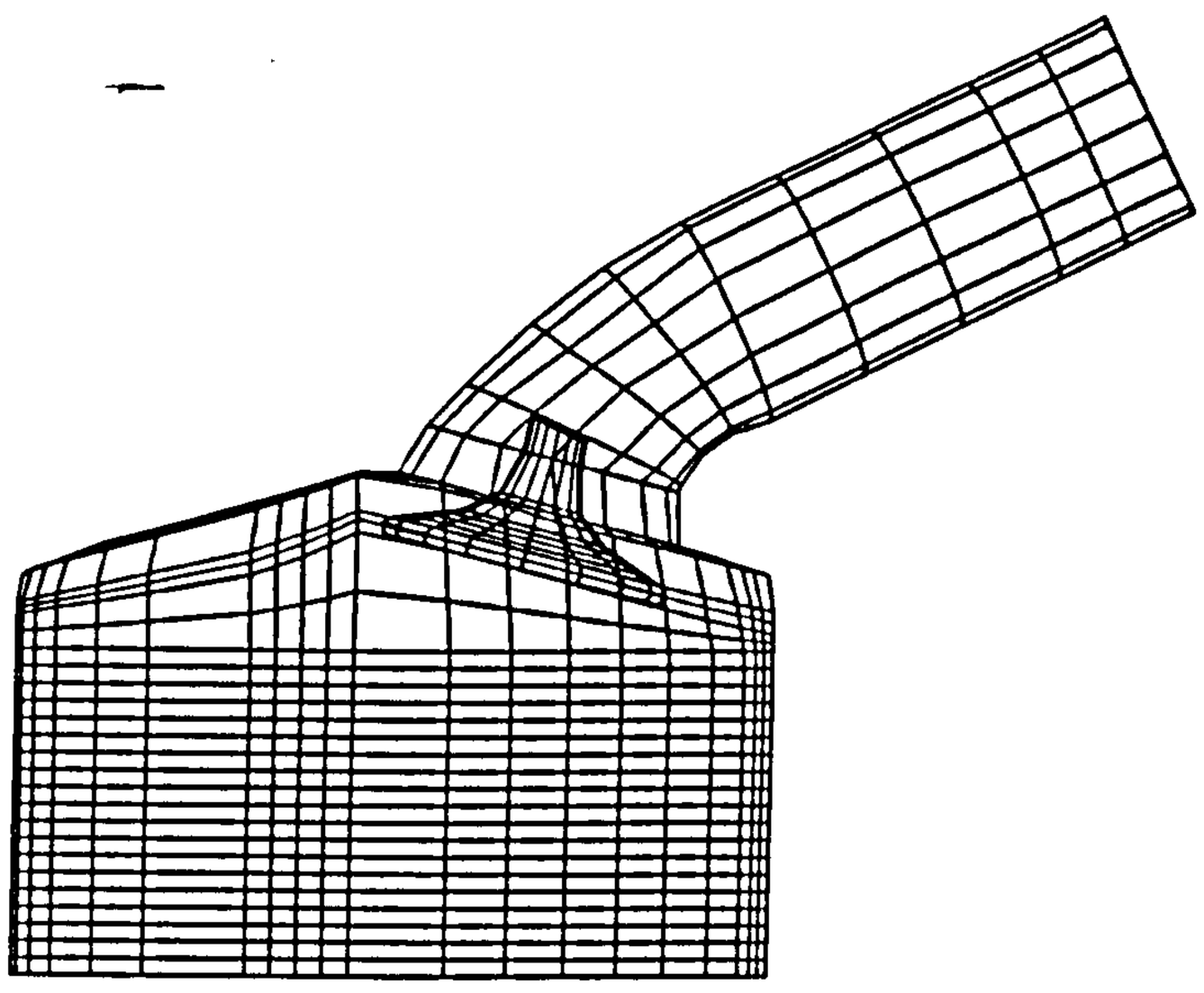


Figure 4.13 Mesh-5 with piston head in middle of piston stroke

The NEWXYZ subroutine was modified to stretch the cells according to the piston's movement through the induction and compression strokes. In the first instance, the cells were stretched from this central position to BDC and then compressed almost to TDC. The cell faces at the top of the inlet port were defined as a pressure boundary for the intake stroke, which was then changed to a wall boundary for the compression stroke.

Valve movement was not attempted for this mesh and so the valve was fixed at a lift of 6mm for this case. This meant that the air could flow back into the port during the compression stroke, and so this transient case is only a demonstration of STAR-CD mesh moving capabilities for the piston in 3D and not a realistic engine simulation.

Figures 4.14 to 4.17 over-leaf show a selection of results at different time steps for the transient run at a cross-section parallel to the symmetry plane and passing through the centre of the valve. The very first time-step before the proper flow field can be established is shown in Figure 4.14, and illustrates the effect of the descending piston. Figure 4.15 shows BDC where a tumbling motion can be seen in the cylinder. The piston ascending is shown in Figure 4.16, and the tumbling motion remaining in the cylinder can clearly be seen. In a realistic simulation, the valve would be closed at this point, and so the flow back up into the port which can be seen in Figure 4.16 would not occur. Finally, Figure 4.17 shows the piston near to TDC at the point where the cells have been compressed to the limit, and it is at this stage that cells would have to be deleted in layers to move the piston boundary to TDC.

Mesh-5 had cells that were large enough to withstand stretching to BDC and compressing to TDC. For a finer mesh, which would be required for a proper transient simulation of the port and cylinder flow, the cells would become too distorted with such a large degree of stretching and compression, and so cell layer deletion and addition would be required.

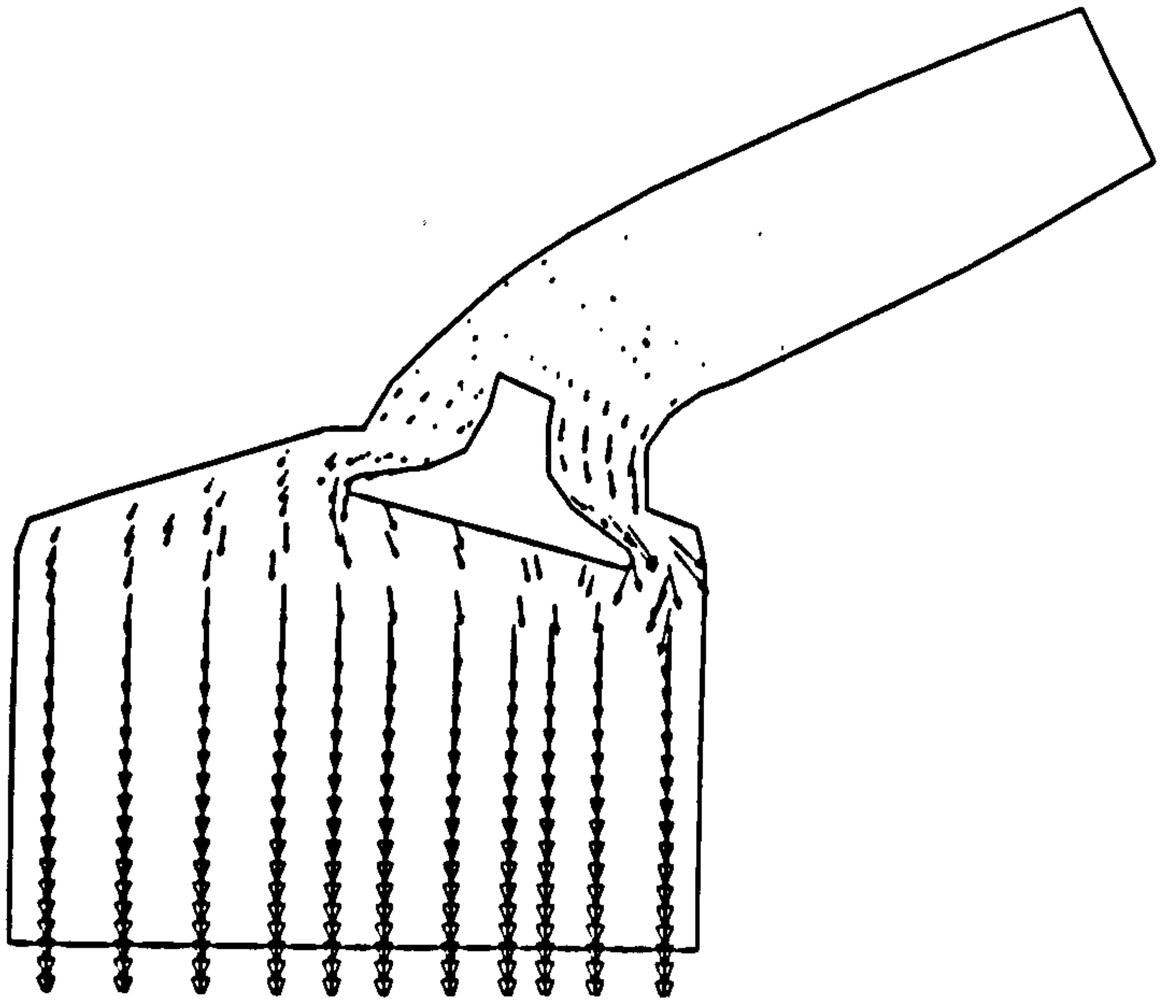


Figure 4.14 Transient analysis - piston descending

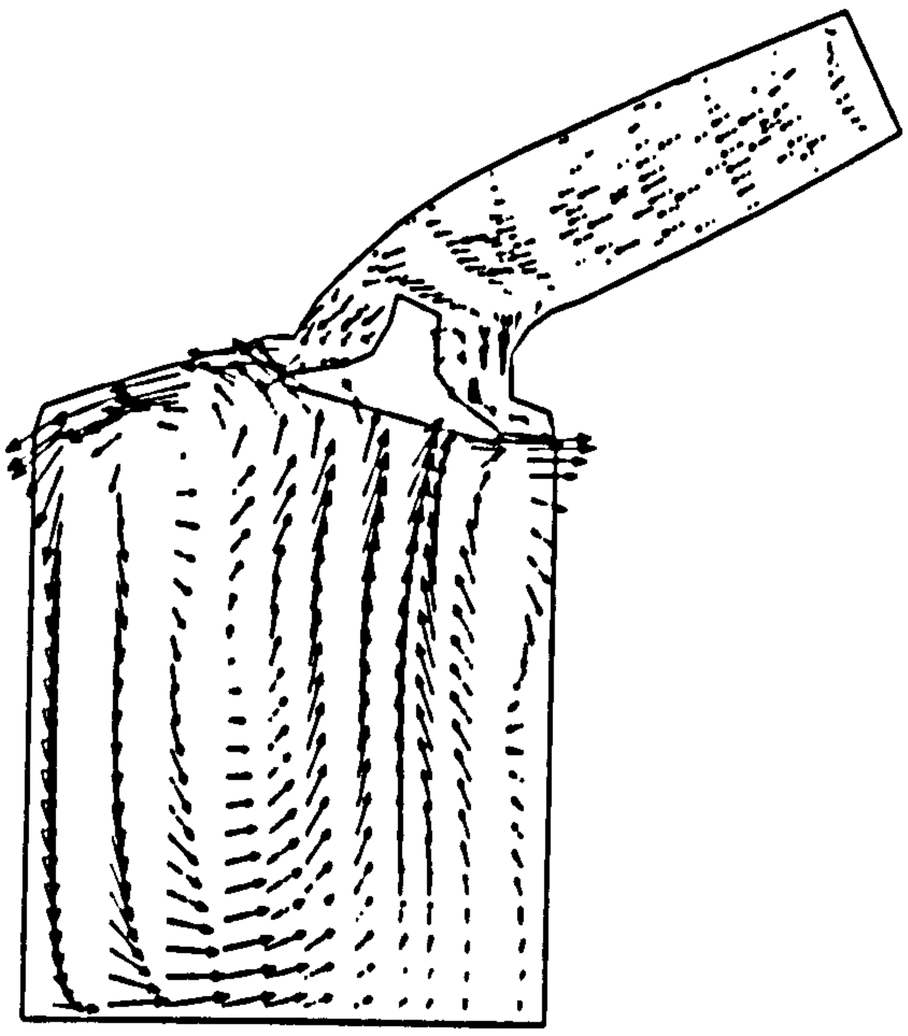


Figure 4.15 Transient analysis - piston at BDC

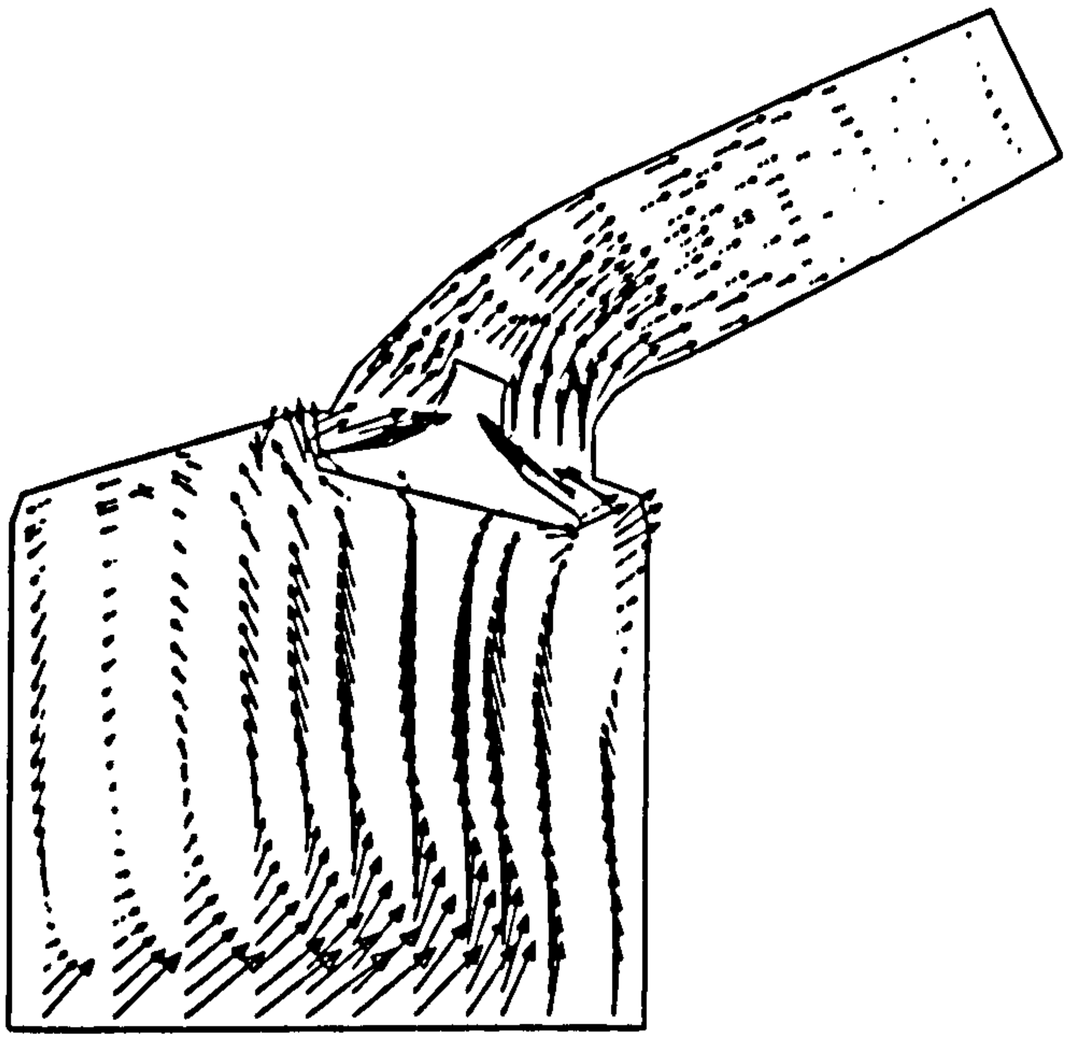


Figure 4.16 Transient analysis - piston ascending

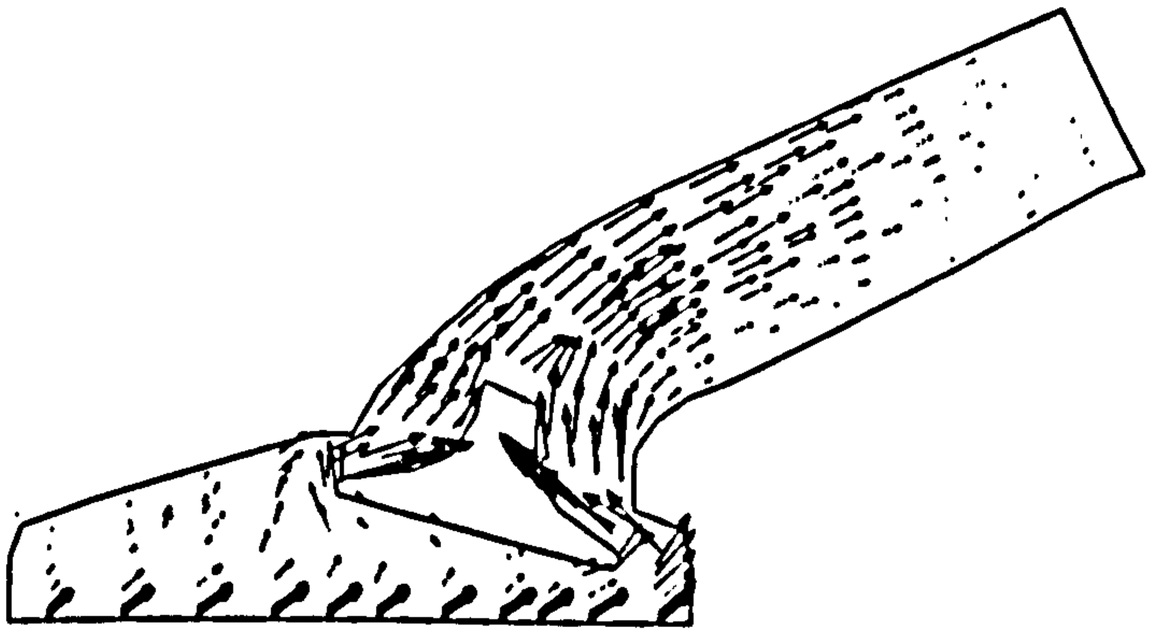


Figure 4.17 Transient analysis - piston nearing TDC

4.8 Using CFD in Jaguar's engine design

The results presented in this Chapter have shown that the some of the CFD predictions using mesh-4 for steady flows correlate quite well with experimental results. These fair results were found even with the disadvantages of using a relatively coarse mesh and an assumption of a symmetric in-cylinder flow field, which proved to be incorrect for the 4mm valve lift case.

Indeed, a flow related geometry fault could be seen from the CFD plots, which indicates that CFD may be a useful tool in the engine design process at Jaguar. The work presented in Chapters 2 and 3 highlights an optimum method for obtaining CFD meshes from Jaguar's current CAD system. In determining this optimum, the mesh quality has seen to be crucial. For example mesh-2 was much faster to generate than mesh-3 or mesh-4, but the cell quality was poor. Indeed CFD runs could only be satisfactorily performed using mesh-3 or mesh-4, and these took longer to create. The relative timescales for mesh generation are shown in Table 4.2.

Mesh	Time to generate	Comments
Mesh-1	5 days	Mesh functionalty removed from I-DEAS
Mesh-2	6 days	Poor quality cells generated. Mesh does not run
Mesh-3	15 days	Cell quality ok.Model does not match experimental rig
Mesh-4	15 days	Cell quality ok. Model matches experimental rig Mesh runs ok in STAR-CD

Table 4.2 Mesh creation times

As was seen in this Chapter, only mesh-3 and mesh-4 produced meshes of sufficient cell quality to be run in STAR-CD. Thus the method of generating meshes by manually creating mapped mesh volumes on CADDS curves and those generated using solid modelling is recommended to Jaguar. In generating mesh-3 and mesh-4, solid modelling proved to be valuable for generating the parts of geometry. This approach is in fact a combination of routes C and D which are shown in Figure 2.2. If solid modelling had not been used to generate the valve, roof and cylinder, that is route C only in Figure 2.2, it is anticipated that the time to create a mesh could have been increased by one week.

In summary, the author has determined a method for generating meshes using the I-DEAS software which is currently used by Jaguar. The technique produces a set of meshes with differing valve lifts within three weeks. These meshes are suitable for running using STAR-CD and have shown to give some fair correlations to experimental data, even using a coarser mesh than would normally be used for such analyses. The author anticipates that a better correlation would be achieved using a finer mesh, although some discrepancy between experimental and predicted results would be expected for areas of highly swirling or separated flow. This is due to the k- ε turbulence model, as discussed in Section 4.5.1, and its effects would be most significant in the separated flow region beneath the valve.

4.8.1 Recommendations to Jaguar

The recommendation to use the most manual of mesh generation techniques examined has implications on the way in which CFD may be integrated into Jaguar's engine design. The options open to Jaguar were discussed in Section 2.10. The recommendation here is to retrain the CAD draughtsmen to create computer models suitable for all aspects of computer modelling, which includes CFD and FE structural analysis. This will minimise the amount of repetition when creating a computer model. It will also make CFD analysis more accessible to the engine developers, since the skills required are less specialised.

Both I-DEAS and STAR-CD have proved to be suitable for the CFD analysis, although less

useful than was originally anticipated. Since the computer software market is continually changing, however, it is strongly recommended that regular software assessments are made. This will ensure that Jaguar uses the tools most suitable to the CFD analysis required. For example, this Chapter has presented the first steps in transient engine flow analysis in which STAR-CD was entirely appropriate. If a further extension includes, say, combustion modelling, another code survey should be done to find the most suitable code for this work.

4.9 Summary of CFD simulations

In this Chapter computational models of a number of different meshes have been run using STAR-CD. Of these, only the mesh generated using mapped mesh areas as described in Chapter 3 were suitable. An initial comparison between CFD results for the latter showed large differences. Subsequent remeshing, including extending the cylinder to match the experimental geometry rather than the actual engine geometry gave much improved correlations. Differences still occurred in the benchmarking, but the general flow trends were well predicted using CFD given the coarseness of the mesh.

Thus for steady flows, using meshes generated in I-DEAS from mapped mesh areas, STAR-CD produced results sufficient to enable engine designers to modify and improve the engine in question. The timescales for analysing a engine geometry using the recommended method of mapped mesh volumes was about four weeks in total, three weeks to create the mesh and one week to perform the analysis. This is just acceptable to Jaguar, but if a computer dedicated to CFD analysis was installed at Jaguar, rather than using the existing multi-user CONVEX, then the analysis time could be reduced by one third.

The timescales for the entire mesh generation and analysis for three valve lifts using the methods proposed in this thesis are at most twelve weeks, which is less than the time required to generate a single mesh using STAR-CD's pre-processor alone.

Further examination of STAR-CD capabilities was made for transient flow analysis, in particular, modelling the moving piston. Moving the valve for a transient analysis was defined in a computer program written by the author. This transient study was made for a 2D case, but the results obtained from 2D give some confidence that a 3D transient analysis can be performed using STAR-CD. Indeed, the first step towards this was presented here by demonstrating a 3D transient simulation at a fixed valve lift with a moving piston.

5. AUTOMATIC MESH GENERATION

5.1 Introduction

As has been seen in Chapter 4, the mesh most suitable for CFD analysis, mesh-4, was created using mesh generation techniques that took the longest time and the most user-skill. Clearly there is a need for a fast and easy method of mesh generation for such CFD applications. The availability of fast automatic mesh generators which produce good quality meshes that are appropriate to the flow situation would significantly increase both the use of CFD and its use-fulness within industry.

The solution algorithm used in the CFD code dictates which cells types can be used in a mesh. Thus the selection of a method for generating the mesh largely depends upon what types of cells may be used. Generally, tetrahedra are easier to generate than hexahedra for all but the simplest of shapes, primarily because the tetrahedra cell faces are triangles each of which defines a plane. Mathematically, this makes tetrahedral mesh generation a simpler process as distinct from quadrilateral faces in which the corner points do not necessarily lie in the same plane. Although CFD solution algorithms such as SIMPLE and PISO were initially developed for use with hexahedral cells, commercial solvers are now available for use with fully tetrahedral meshes, such as the finite volume code RAMPANT and the finite element code FIDAP. STAR-CD accepts hexahedral, prismatic, pyramidal and hexahedral cells and combinations of these. Since a suitable solver can usually be found to use a mesh with a mix of cell types, we are free to select the most suitable methods of mesh generation available.

This Section therefore describes types of mesh generation methods commonly used and the degree of automation which can be expected with each technique. Section 5.2 outlines each of these types of mesh generator and describes how these techniques may be combined to create meshes of complex geometries and how they are used within current commercial software. Common problems facing methods of automatic mesh generation are discussed in Section 5.3, and commercial automatic mesh generators are outlined in Section 5.4. Finally,

Section 5.5 gives a summary of the Chapter.

5.2 Mesh generation and automation

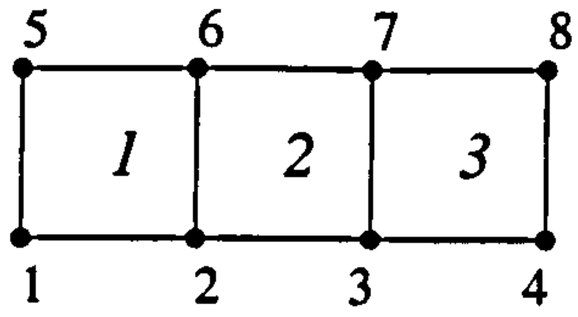
Techniques for generating meshes can broadly be grouped into seven different categories as described by George [63]. These groups are

1. explicit mesh definition
2. mapping functions
3. solution of PDEs
4. deformation or modification of a simple mesh
5. composition of meshes obtained by functions (mapping or PDEs)
6. derivations from boundary data cell by cell
7. composition of meshes obtained by any of the above methods

These methods are briefly described in the following subsections, but methods 2 and 3 will be considered together as mapping functions and methods 5 and 7 will be considered together as compositions of other meshes. These groupings merely link together similar mesh generation methods for simplicity. The subsections below also detail which cells can be generated using each technique, along with a broad description of how automated each technique may become. It must be noted, however, that this is not an exhaustive treatment of every mesh generation method devised, merely an overview of some of the most common methods used.

5.2.1 Explicit mesh definition

The most basic of the mesh generation techniques is explicit mesh definition. First a set of vertices are created at certain positions within the flow domain. Each vertex is assigned a unique label and the vertices are usually numbered sequentially. Individual cells are then defined by listing the vertices at each corner of the cell as illustrated in Figure 5.1 for a 2D case.



Cell	Vertices
1	1 2 6 5
2	2 3 7 6
3	3 4 8 7

Figure 5.1 Simple mesh generated explicitly

This method is most suited to quadrilateral cells in 2D and hexahedral cells in 3D, since lines and planes in the mesh are easy to visualise and to plan out. Tetrahedral cells are particularly unsuited to this method, not because the technique cannot be used, but simply because natural planes in the structure of a group of adjacent tetrahedra do not necessarily exist. This means that the meshes are very difficult to visualise and to plan.

Using local coordinate systems and different coordinate systems such as Cartesian and polar, a mesh can be built up in sections. This reduces the effort in creating a mesh explicitly. Further techniques to speed up mesh generation use patterns and groups of vertices and cells which are then copied, rotated and reflected to different sections of the flow domain. These techniques make this method straight-forward for regular and simple geometries, as demonstrated by Cook [64] and by Gordan and Hall [65]. Such generation of groups of vertices and cells is about as far as explicit mesh generation can be automated. For large and complex meshes this method becomes unwieldy and impractical, and so other methods of mesh generation must be used.

5.2.2 Mapping Functions

A mesh of a simple geometry can be generated explicitly as described in Section 5.2.1. This mesh can then be mapped onto a more complex geometry using any number of mapping functions. The simplest mapping is direct one-to-one mapping, where boundaries in the simple geometry correspond directly and proportionally to the boundaries in the complex geometry.

Internal vertices are then mapped to the geometry by interpolation from the boundary vertices, as illustrated simply in Figure 5.2. One example of this method is documented by Zienkiewicz and Phillips [66].

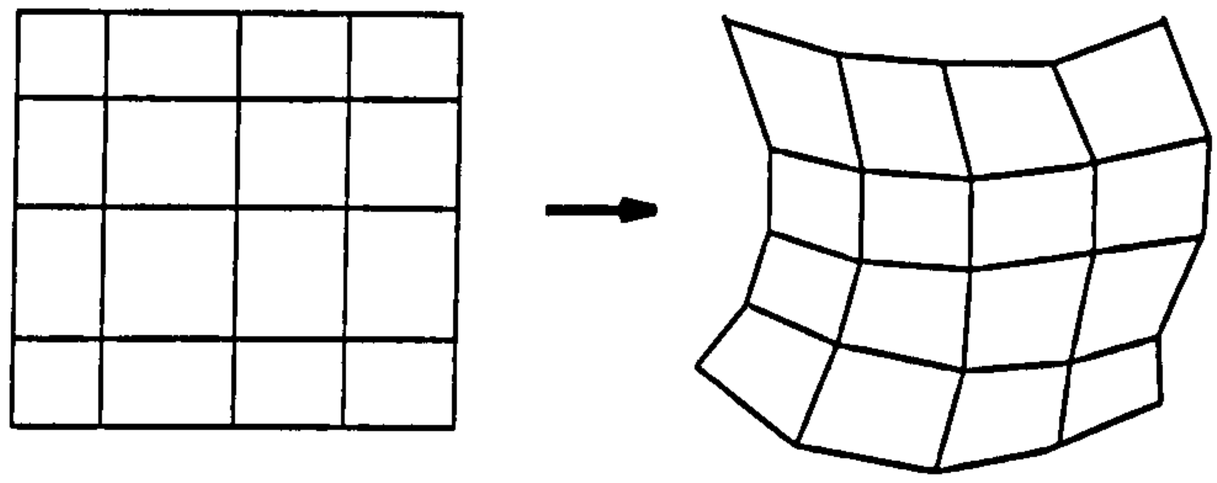


Figure 5.2 Mapping a mesh onto a more complex geometry

Another common use of mapping is projecting a 2D planar mesh onto a curved surface. This approach cannot work where the geometry curves through 180^o, however, since the cells on the 2D plane could not be mapped onto a surface normal to it. This limitation therefore makes the method unsuitable as a general purpose technique [67].

The mapping functions used depend upon the geometries involved, but the user can define any mapping function that is suitable. Further developments include the definition of mapping functions by solving PDEs for a mesh parameter such as cell density, or even flow variables from a previous CFD solution. One common example of this is called 'smoothing' where the cell quality defined by internal angles is used to determine new vertex positions.

Mapping can be used for any 2D or 3D cell type, since it is only the vertices which are transformed by the mapping function whilst the cell connectivity remains the same. The resultant cell quality must be monitored, however, because the mapping often distorts the cell shape. Mapping a mesh from a simple geometry to a more complex one is done numerically, and so the user is less directly involved in the generation of the final mesh than they would be

if the mesh were explicitly defined. The user must still generate the initial mesh, however, and decide upon a suitable mapping function or mesh parameter to use. Mapping functions can therefore be used effectively in a number of real engineering situations, but may still require significant user input.

5.2.3 Mesh modification

A simple mesh with quadrilateral or hexahedral cells is first generated using any simple technique such as explicit mesh generation, mapping functions or interpolation as described above. The mesh modification technique is then used, as illustrated in Figure 5.3 below.

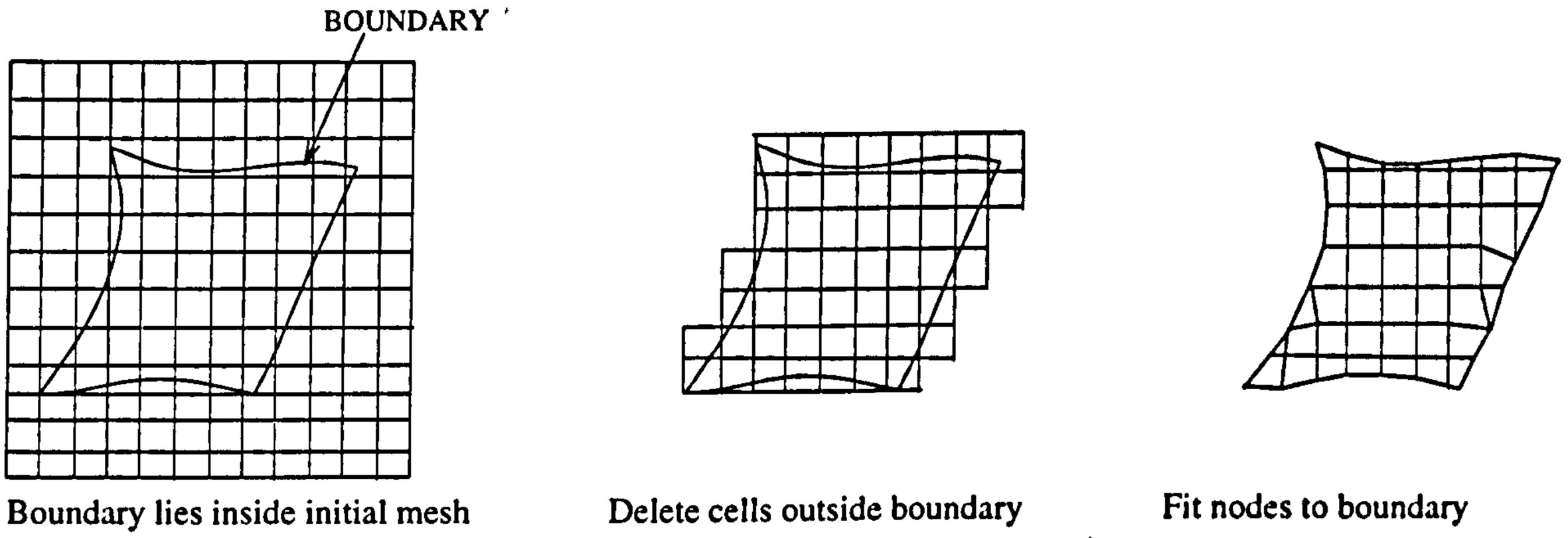
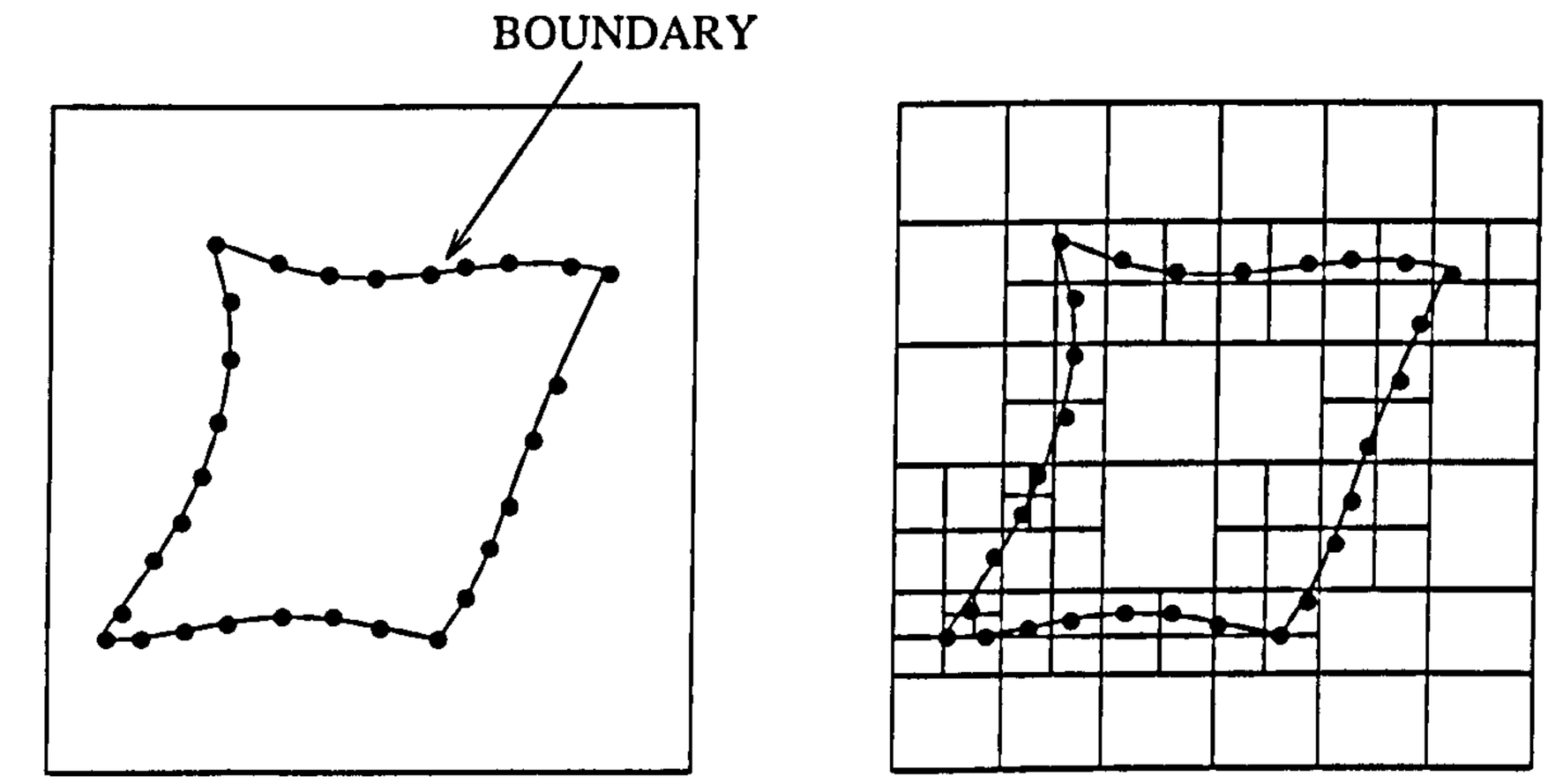


Figure 5.3 Stages in mesh modification

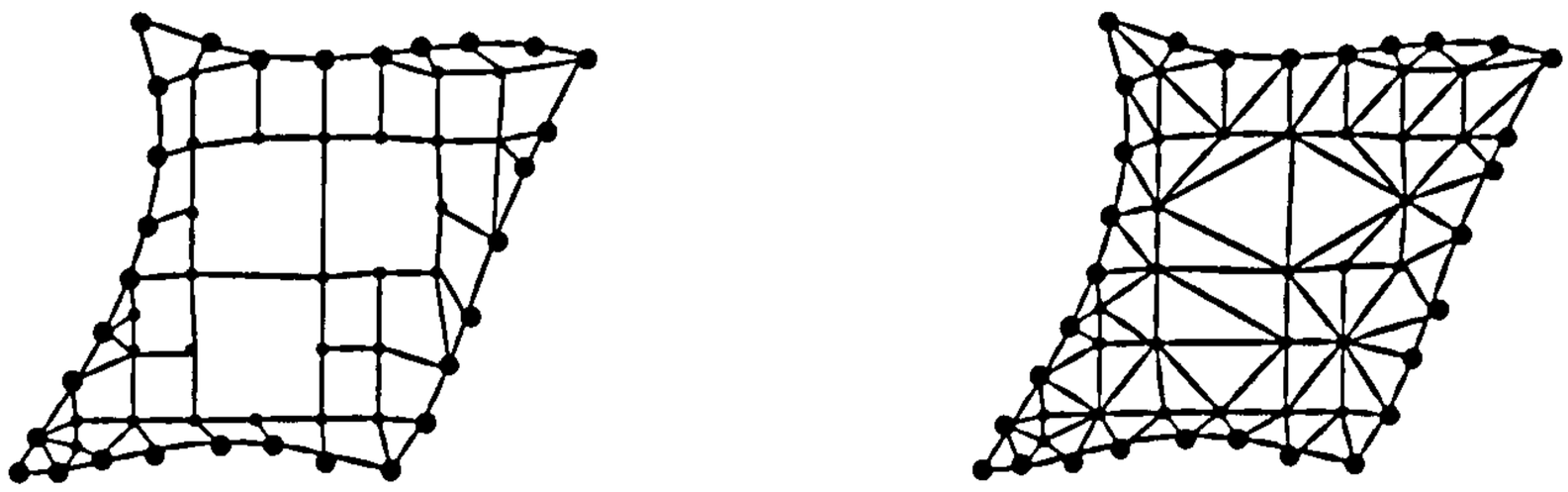
The boundaries of the flow domain lie entirely within this simple mesh, defined as a set of curves or surfaces. Each cell is tested and if it lies wholly outside of the domain, then that cell is deleted from the mesh. This leaves cells either within or crossing the boundaries of the flow domain. The cells crossing the boundaries are then modified to fit the boundaries either by moving appropriate vertices or by splitting the cells.

A variant of this process is called quadtree in 2D or octree in 3D as described by Yerry and Shephard [68]. In 2D, a single quadrilateral is generated which contains the entire flow domain, and the latter is described by a set of contour points and edges, see Figure 5.4(a). The quadrilateral cell is split into four smaller quadrilaterals and each of these new cells is then tested to determine which contour points it contains. Where a cell contains a contour edge, it is divided into four more cells. This process continues until no cells exist that contain more than one contour point. This is illustrated in Figure 5.4(b).



a) Single cell surrounds boundary

b) Cell subdivided



c) Nodes fitted to boundary

d) Triangular mesh generated

Figure 5.4 The quadtree method

As before, any cells lying outside the domain are removed from the mesh and the cells lying along the boundaries are split of have their vertices moved onto the boundary, Figure 5.4(c). Since the cell faces no longer match one to one, each cell is split into triangles so that the resultant mesh has matching faces, as shown in Figure 5.4(d). Finally the cell quality can be improved by smoothing the mesh.

These cells in the quadtree mesh are listed in an hierarchical tree structure to identify both them and their parent cells, as illustrated in Figure 5.5.

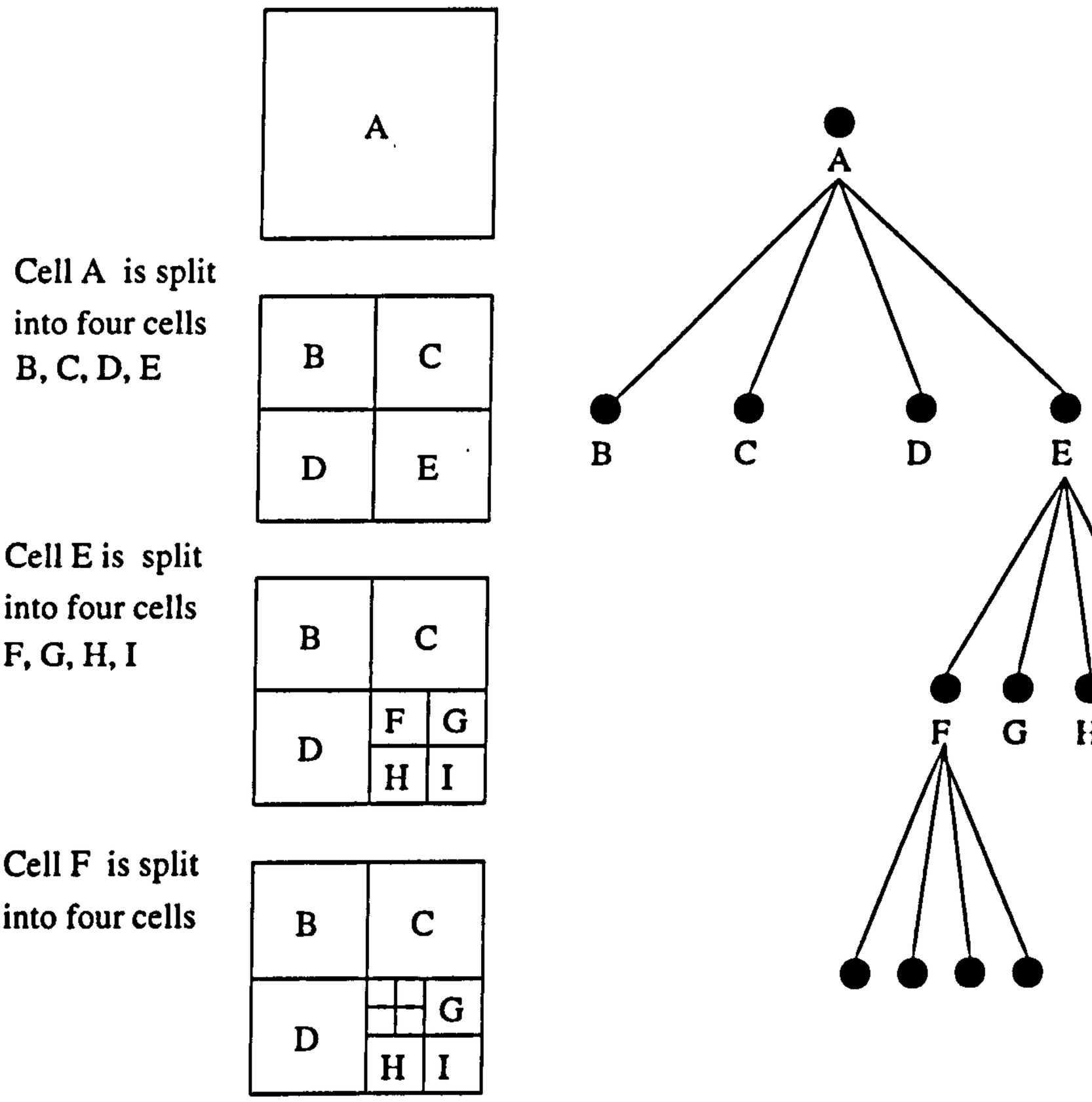


Figure 5.5 Tree structure for quadtree method

This mesh generation method can have a large degree of automation. The user must generate the initial mesh for mesh modification, but this can be done simply by enclosing the domain in a mesh generated using interpolation. Rules for splitting the cells must be devised, but once programmed into a computer, they can be used without further intervention by the user. The boundaries must of course be specified in a way that allows the use of an automatic method for determining whether points lie inside or outside of the domain. Indeed, the boundary definition is critical to quadtree and octree meshing because vertices that are too close together can lead to an excessive number of cells generated.

5.2.4 Mesh derivation from boundaries

There are two main methods for deriving meshes cell by cell from boundaries, and these are the advancing front method, described by Peraire et al [69] and the Delaunay method as outlined by Baker [70]. Both these methods are highly suitable for generating triangular meshes in 2D and tetrahedral meshes in 3D. For clarity, the description of both methods is done for the 2D case in the following subsections.

5.2.4.1 Advancing front method

The boundary of the domain is discretised into lines in 2D. Each of these 'boundary cells' forms the base for a new cell which extends into the domain. Cells are created by linking a boundary cell to a vertex which lies inside the domain. This vertex can be found from a set of points pre-positioned in the domain or alternatively the vertex can be generated at an appropriate position with respect to the boundary cell. In each case, the vertex is selected according to proscribed rules based upon cell quality. This technique is illustrated in Figure 5.6.

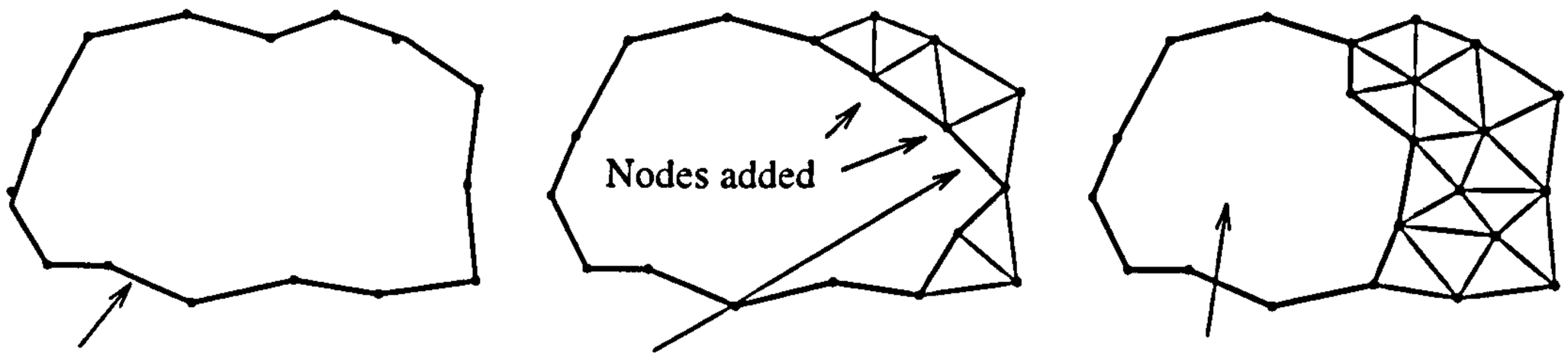


Figure 5.6 Advancing front technique

It can be seen in this Figure that a front of cell faces advances across the flow domain, each time forming a polygon that must be filled with cells in order to complete the mesh. Whilst the polygon cannot always be subdivided into a set of quadrilaterals, it can always be filled by a set of triangular cells, and so this method is ideally suited to triangles and to tetrahedra in 3D.

Again this is essentially an automatic technique, but the definition and discretisation of the boundaries and the point placement within the domain (where appropriate) may well be a significant task for the user.

5.2.4.2 Delaunay method

This method consists of successively subdividing a one cell mesh by inserting new points one at a time and locally refining the mesh near the point. For the 2D case, a set of points in a plane are to have a mesh of triangular cells fitted between them. First a large triangle is created that completely encloses all the points. The points are incorporated into the mesh one by one and new triangles are formed in the following way;

Every triangle has a unique 'circumcircle',that is, a circle which passes through each of its corner points. The circumcircle is found for every triangle in the mesh, see Figure 5.7, and if

the inserted point lies within a triangle's circumcircle then that triangle is deleted as shown in Figure 5.8

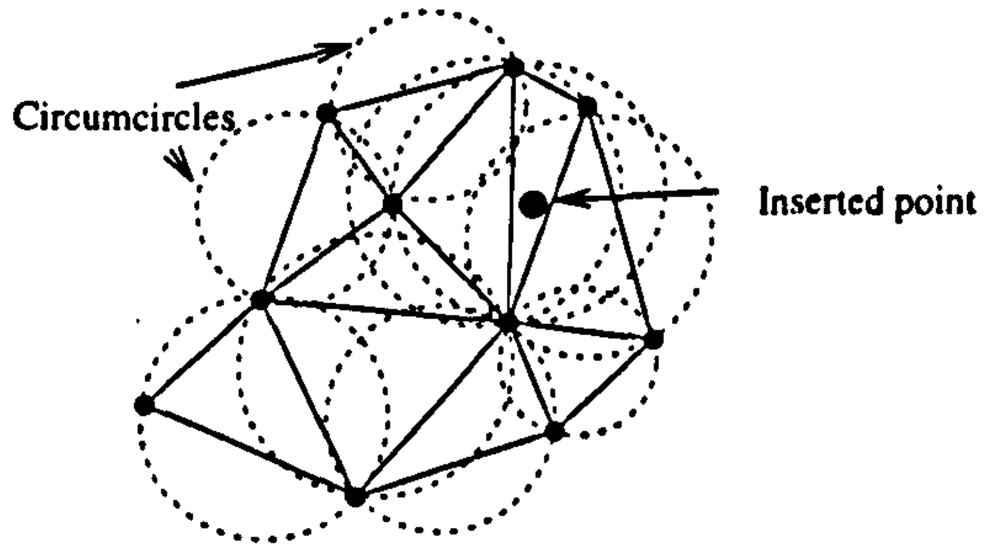


Figure 5.7 Circumcircles which enclose inserted point

Once all the relevant triangles in the mesh have been deleted, a polygon then surrounds the inserted point.

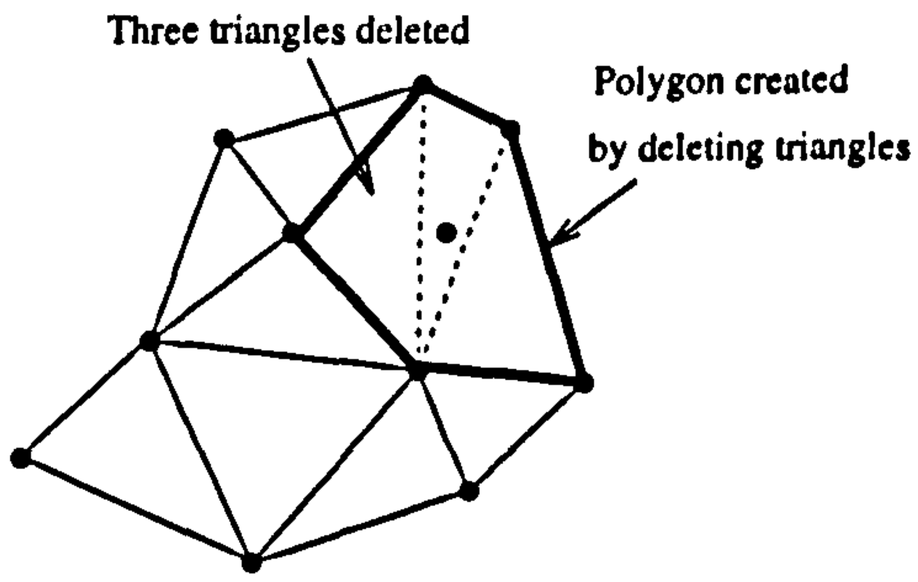


Figure 5.8 Deleted triangles leave polygon

The inserted point is joined to each vertex in the polygon creating a new set of triangles, as shown in Figure 5.9.

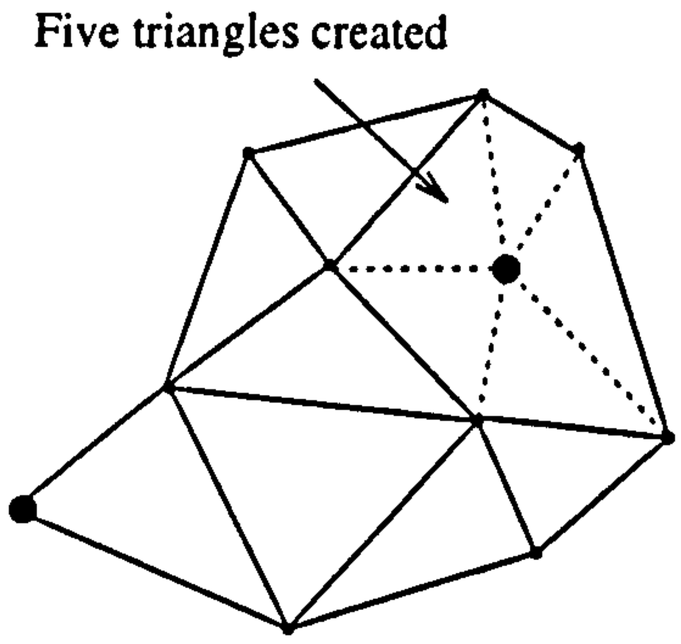


Figure 5.9 New triangles created

After each point has been inserted, the mesh fills the larger triangular region without any gaps, and so the final mesh achieved mesh will also fill the entire region without any gaps. The process is repeated until all the points have been inserted. Once all the points have been inserted, any triangles connected to the vertices of the original, single triangle will be deleted. This leaves a mesh of triangles fitted through the set of points.

One major problem with the basic Delaunay method has been the inability to proscribe the surface boundaries with cells rather than points. Thus the mesh generated will not necessarily conform to the boundaries of the flow domain, as illustrated in Figure 5.10.

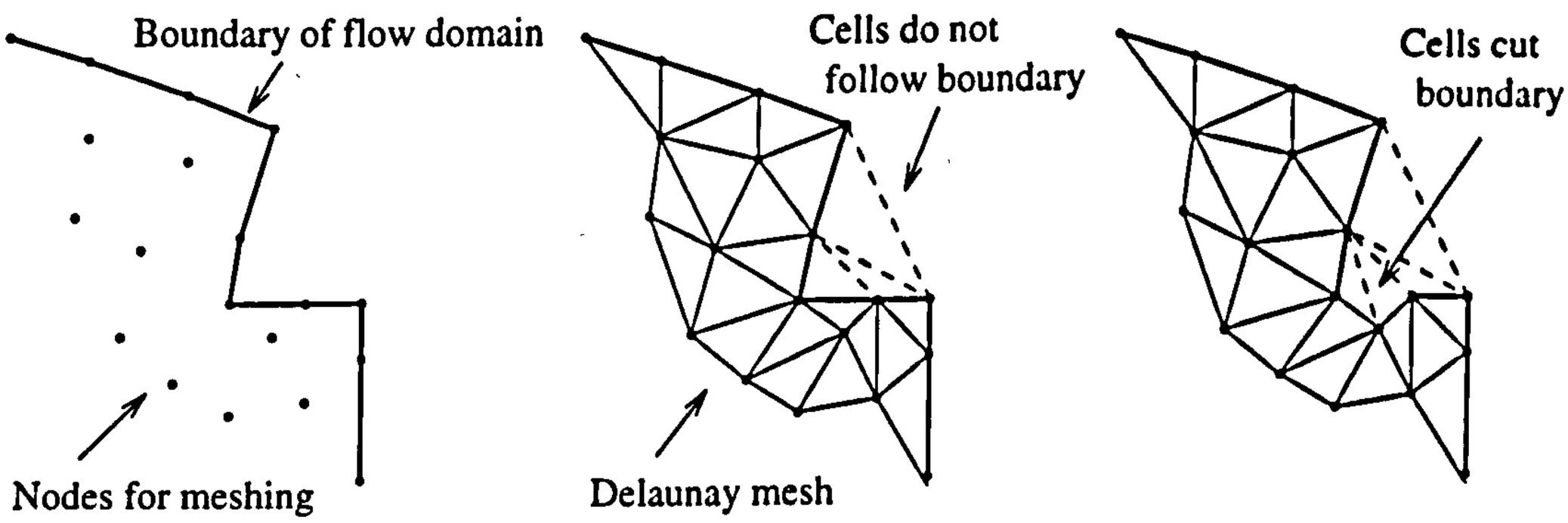


Figure 5.10 Delaunay mesh mis-aligned with boundary of the flow domain

Maksymiuk and Merriam [71] propose a method for 'pruning' those tetrahedra which do not follow the boundary, although this is not a robust method for all surfaces, whereas George et al [72] report a method for edge-swapping to improve surface discrepancies.

This method has good scope for automation since the rules of finding the correct circumspheres, deleting them and inserting the new point can be programmed into a computer. However point placement within the flow domain may still require a lot of user effort.

5.2.5 Combination methods of mesh generation

Different methods of mesh generation may be combined to create more powerful techniques. One such method is the 'multi-block' method in which the flow domain is divided into a set of sub-domains. These sub-domains are of simpler topology than the entire domain in which meshes can be generated more easily. Indeed, different topologies of these sub-domains can be meshed in different ways, and a set of rules can determine which topology the sub-domain is and therefore what method of mesh generation is to be used. In I-DEAS, the sub-domains, called mesh volumes, are created manually by the user. Mesh generation within the mesh volumes is wholly user defined using interpolation for hexahedra in mapped mesh volumes.

Multi-block methods can be further automated by medial-surface subdivision [73] where points are placed on the surface of the domain and a Delaunay mesh is generated using only these points. This tetrahedral mesh then enables the domain to be divided into sub-domains according to a set of predefined rules. If the sub-domains have one of a number of topologies defined in the mesh generation program, each of which has a predefined method of mesh generation, then the complete mesh can be generated. This method will generate meshes with either tetrahedral or hexahedral cells.

Another combination of methods is the use of octree meshes to generate a set of vertex points within the domain. This vertex set can then be used with both the advancing front method, as described by Jin and Tanner [74], and with Delaunay methods. In such cases, the knowledge

of neighbouring cells and vertices can significantly speed up the meshing by reducing the searching of every boundary cell or circumsphere in the mesh.

5.3 Problems in automatic mesh generation

The mesh generators described in Section 5.2 may have their individual problems and limitations, some of which can be circumvented. They all share some common problems, however, since to be fully automatic the mesh generators must take into account both the description of the geometry defining the flow domain and the physics of the flow field itself.

5.3.1 Description of the domain

CAD data has great potential for use with automatic mesh generation in a CAE environment. Complications may arise, however, if fully automatic mesh generation is required, that is, if no user input is necessary.

For example, mapping functions may require a mathematical description of a surface, but often engineering surfaces are not or cannot be described mathematically. Instead, CAD surfaces may consist of a set of facets or patches. Whilst vertices may be mapped onto these facets using projection techniques, more complex mathematical mapping functions cannot use these surfaces.

CAD data is often produced to replace detail engineering drawings or to direct NC cutting machinery. Complete drawings will contain details such as bolt holes and the bolt, say. Such tiny details may be insignificant to the flow field and so unnecessary in CFD calculations. Indeed, fine details can often cause problems when using automatic mesh generators since an excessively fine mesh may be generated around the small feature, increasing the number of cells required. This small detail may also cause a breakdown of the mesh generator itself by the creation of cells which may have vertices closer than the required point coincidence tolerance.

CAD drawings for NC machinery do not have to be as rigourously defined as the geometry used for CFD. CAD surfaces may overlap one another, or two surfaces may adjoin another surface along the same curve. Whilst this is no problem for machines cutting one surface at a time, a mesh generator trying to create a coherent mesh requires the domain to be completely closed with no gaps or overlaps between surfaces.

5.3.2 Local Mesh Refinement

CFD has a requirement for a mesh to be fine enough to capture the salient features of a flow at a scale where the results are independent of further mesh refinement. Grid independence, as it is called, is not always a viable goal in industry since the analysis would take too long, require too much computing power and hence cost too much.

One work around for this problem is to use adaptive meshing techniques as described in Section 3.6. Adaptive meshing itself could be automated by programming a computer with CFD experience and create an 'expert system' to determine what the mesh density should be. If the mesh were automatically refined also, then an automatic mesh generator could work in conjunction with the solver.

This approach would significantly reduce the amount of user input. It is a trial and error approach to mesh generation, however, which depends upon many analysis runs to create an acceptable mesh before the run producing the eventual results is performed. This could conceivably take longer to analyse a given flow condition than a single mesh generated by an experienced CFD analyst.

5.4 Commercial automatic mesh generators

There are currently a number of automatic mesh generators available commercially. These software packages use CAD data as the basis for creating the geometry. A problem shared by all commercial mesh generators is the lack of an effective, industry-standard CAD data

interpreter which would allow a CFD mesh to be built immediately without first having to modify the CAD data.

Some general CAE packages such as I-DEAS and PATRAN have automatic meshing capabilities. In I-DEAS users create their own mesh volumes and define the number, type and distribution of cells in each mesh volume. More automatic is I-DEAS' single mesh-from-object command but the geometry first has to be created as a solid model within I-DEAS, which limits the CAD interface unless solid modelling is used in all the CAE applications for that geometry. The mesh is then automatically generated from the solid creating quasi-2D cells on the surface initially, and then creating tetrahedral cells throughout the solid.

Another commercial mesh generator is the Tgrid module in RAMPANT which uses the Delaunay method to generate a tetrahedral mesh. Whilst good quality cells may be achieved, the boundaries must have a very good quality triangular mesh which can often require quite a lot of user intervention in the mesh generation itself. The triangular cells can either be generated by RAMPANT's own surface mesh generator, or by using other software packages such as I-DEAS, PATRAN or ICEM-CFD.

Full integration of CAD data into an automatic mesh generator has yet to be achieved and so a fully automatic mesh generator useful to someone inexperienced in CFD is still not available.

5.5 Summary of chapter 5

In this Chapter, different types of mesh generation were examined, particularly with respect to creating a fully automatic mesh generator which has no user input. Common problems for all the methods used are firstly the inability to use CAD data directly as it has been created for other CAE applications. Secondly, the mesh must be refined according to the flow conditions. Whilst an experienced CFD analyst may create a mesh that will produce good results first time around, automatic mesh generators must use successive solution runs and adaptive

meshing in order to produce a mesh which will give good results.

Commercial mesh generators were examined, but no current mesh generator could be described as fully automatic. The next chapter describes the author's own, novel approach to speeding up the use of existing mesh generators in an automatic way.

6. MESH CONSTRUCTION: THEORY

6.1 Introduction

This chapter describes a novel method of mesh generation designed and programmed by the author. The method automatically builds up an unstructured mesh of a complex geometry from meshes of simpler geometries. This 'mesh construction' program includes linking one arbitrary mesh to another and also cutting one mesh from another.

Section 6.2 outlines the philosophy behind mesh construction and Section 6.3 gives an overview of the techniques used. Each of the stages in mesh construction is presented in Sections 6.4 to 6.7 in terms of joining two meshes together. A similar approach is used for cutting one mesh from another, as described in Section 6.8. A discussion in Section 6.9 puts the mesh construction theory in the context of other similar and current work. A summary of this chapter is then given in Section 6.10.

6.2 Philosophy

A complex geometry can often be broken down into elementary shapes such as cylinders and blocks. These shapes are generally quick and easy to mesh using unsophisticated mesh generation techniques such as explicit generation and mapping. In light of this, the author has devised a novel method for automatically constructing one coherent mesh of a complex geometry from meshes of simpler shapes. The aim was to make mesh generation easier and less time consuming for the user.

The simple meshes used in mesh construction could be stored to form a whole library of meshes, ready to be linked in any fashion to form a mesh of a complex geometry. This method of mesh construction could also reflect the manufacture of the required geometry, since meshes could be generated defining the flow region through individual components. Each component mesh need only be generated once and stored in the library. This will make

systematic analysis of new concepts easy to perform, comparing the new component against existing ones.

Another advantage of mesh construction is that meshes with different global cell sizes can be linked. This reduces the overall size of the mesh required for the analysis by restricting the finest mesh to regions where it is most needed. Such variations in local cell size are more difficult to achieve with structured meshes, for example, where a localised high mesh density may be propagated throughout the entire mesh. Also, when engineers modify the geometry in a local area, a new simple mesh of this region could be generated and relinked into the rest of the mesh without having to remesh the entire complex geometry, which is the case for 'one step' automatic mesh generators.

6.3 Overview of mesh construction

Mesh construction can be divided into four stages;

1. cell deletion
2. shell construction
3. cell creation within the shell
4. mesh assembly

In this process, groups of cells and vertices are created and manipulated. Figure 6.1 is a 2D illustration of the process which shows each stage in the process along with the groups of cells and/or vertices that are used in each stage. The suffixed numbers refer to the original meshes used, either MESH1 or MESH2. The groups shown in Figure 6.1 will be referred to throughout this chapter and may consist of cells, the vertices attached to them, or both.

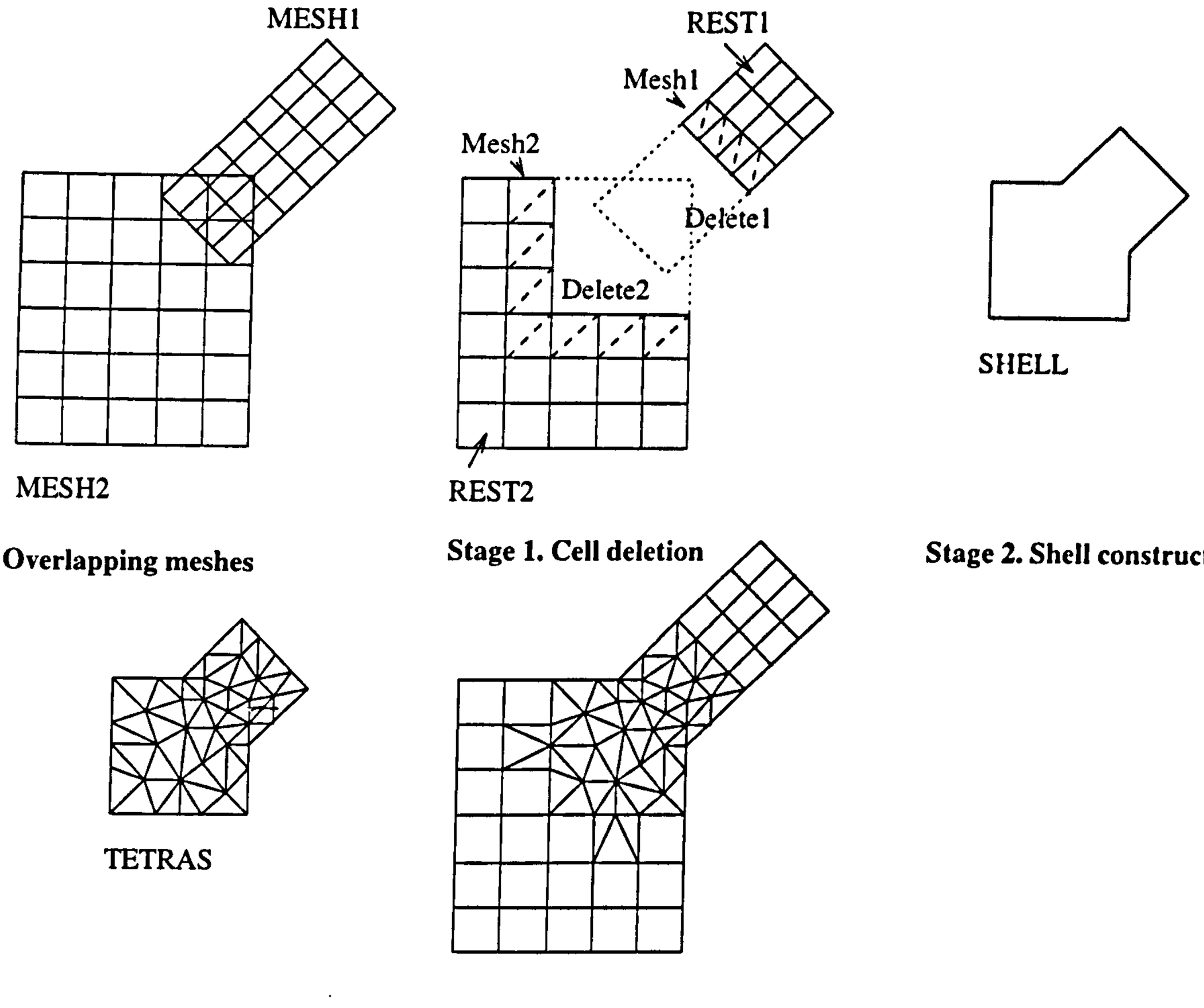


Figure 6.1 Outline of mesh construction

The two original meshes considered are MESH1 and MESH2. Firstly, groups of cells, Delete1 and Delete2, are removed from MESH1 and MESH2 respectively in the region where these meshes overlap. In the second stage, a closed shell of cell faces, 'SHELL', is created from the surface of the deleted elements. Next a mesh of tetrahedral cells, TETRAS, is generated within SHELL. All the cells are then assembled into one mesh in the fourth stage, and the tetrahedral cells are blended into the two original meshes.

Sections 6.4 to 6.7 will now describe each of these stages in detail.

6.4 Cell deletion

This first stage in mesh construction involves finding those cells from MESH1 and MESH2 which overlap one another. It is important that meshes to be linked together actually overlap one another rather than merely touch. This is due to considerations of point coincidence tolerance similar to those discussed in Section 2.7.1 for the facets used in the Boolean operations.

The search for overlapping cells involves testing each vertex in MESH1 against each vertex in MESH2. The number of computations that this requires is proportional to the square of the number of vertices involved. In order to make mesh construction faster, it is therefore helpful to reduce the number of cells considered by first defining a 'working region'. This working region contains a subset of MESH1 and MESH2 in the region where these meshes overlap. A search of the cells in the working region is then made to determine which cells in the two meshes overlap. These two procedures are described in the following sub-sections.

6.4.1 Defining a working region

The working region may be defined either by the user or by an automatic search routine.

User selection of the working region is done by simply entering the maximum and minimum coordinates of the region of mesh overlap, with a little allowance for cells lying just outside this region. It is often a simple task to visually determine the extent of the working region and so it can be a quick method of assessment. Where it is difficult or more complex to define the working region by eye automatic selection can be used.

Automatic selection of the working region can be done by considering MESH1 and MESH2 in turn. A search is made of every vertex in each mesh to find the maximum and minimum coordinate values in the Cartesian coordinate system for that mesh. An overlap region can be defined as the volume enclosed by the overlapping limits of both meshes, as shown in 2D in Figure 6.2. The working region, however, needs to include the cells immediately next to the

overlap region in order to properly blend the tetrahedral mesh to the rest of the mesh in stage 4. The volume enclosed by the overlap region must therefore be increased to create a suitable working region. The overlap region is increased by expanding the maxima and minima by about 2 cell lengths, found for each Mesh by averaging the lengths of cells inside the overlap area. These new coordinate limits then define the working region.

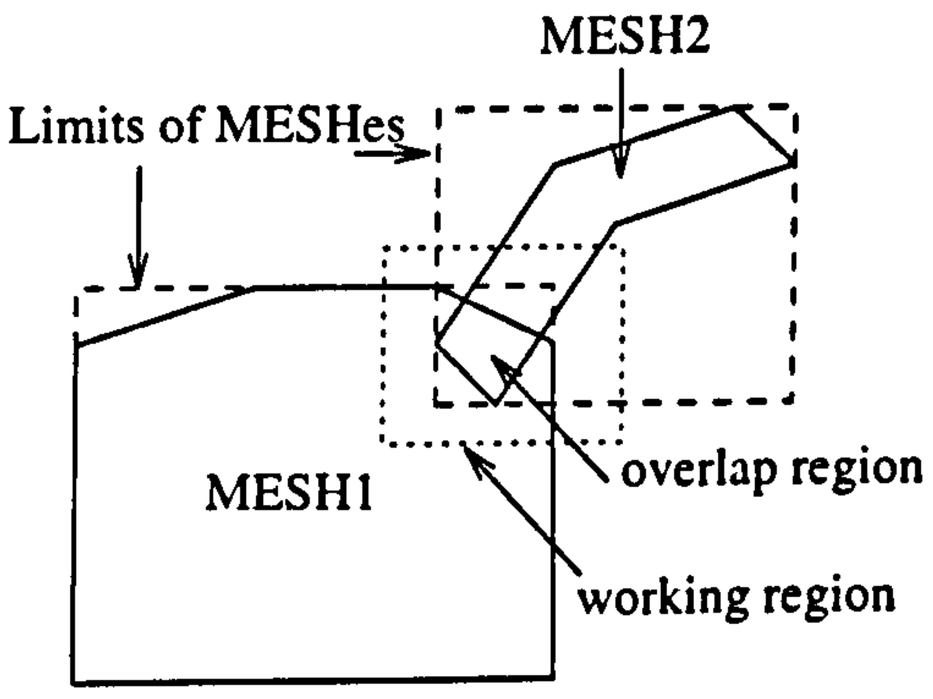


Figure 6.2 Working region using automatic search

This automatic search can give a cruder approximation of the region of mesh overlap than one determined by inspection, but further refinements to this search routine could significantly increase the computational time and effort needed to define the working region. If the meshes are not very large, say, less than 50,000 cells, the use of a working region will not have any significant effect upon the running times for mesh construction, and so the working region can be manually set to include both entire meshes. Alternatively, if a high-powered computer is used it may not be necessary to define a working region at all.

Once the working region has been determined, a search is done on the coordinates of each vertex in MESH1 and MESH2. Referring to Figure 6.1, cells whose vertices all lie inside the working region are listed in Mesh1 and Mesh2 accordingly. The remaining cells are listed in REST1 and REST2.

6.4.2 Selection of overlapping cells

The cells which overlap between the two original meshes are subsets of Mesh1 and Mesh2. All the vertices in Mesh2 are tested to see if they lie within a 'search-distance' of the vertices in Mesh1. This search-distance must be supplied by the user and its influence is described in chapter 7 when test cases for mesh construction are presented.

Basically, any vertices in Mesh2 that lie within the search-distance of the vertices in Mesh1 are listed in Delete2. Cells in Mesh2 that contain any of the vertices in Delete2 are also written to Delete2. The same process is performed to find the vertices in Mesh1 which lie within the search-distance of the vertices in Mesh2. These vertices and their associated cells are then listed in Delete1.

6.5 Shell construction

Before the void between Mesh1 and Mesh2 can be filled with tetrahedral cells in stage 3 of mesh construction, the boundary faces of the void must be defined.

Boundary faces of a group of cells can be found by first listing the faces for every cell in that group. Faces that are shared by two adjacent cells will be listed twice; once for each of the adjacent cells. This means that every face which lies inside a group of cells will be listed twice. It follows that any face listed only once will only have one associated cell and hence lie on the surface of the cell group. Hence the boundary faces of any group of cells can be found in this manner.

This technique is used to find the boundary faces of Delete1 and Delete2, and these boundary faces are then written to Faces1 or Faces2 as appropriate. Any quadrilateral faces are subdivided into two triangles along their shortest diagonal, which ensures the best shaped triangles are formed for any given quadrilateral. The new triangles replace their parent quadrilateral face in the Faces sets which therefore form two closed shells of triangular faces. In order to

link the two meshes together, these two shells must be merged.

Each face in Faces1 is tested against every face in Faces2. If two faces intersect, they will be subdivided by the intersection points found. The new triangles are written to Faces1 or Faces2 as appropriate, replacing their parent triangles. The process continues until no new intersections are found. Thus the two Faces sets are linked together along a mutual line of intersection. This process can be illustrated in Figure 6.3 for a 2D case. Now the two shells are completely interlinked and they can be split into two different sets of faces; those lying on the outside of the entire set of faces, and those lying inside. By deleting the internal faces, a linked set of external faces is left.

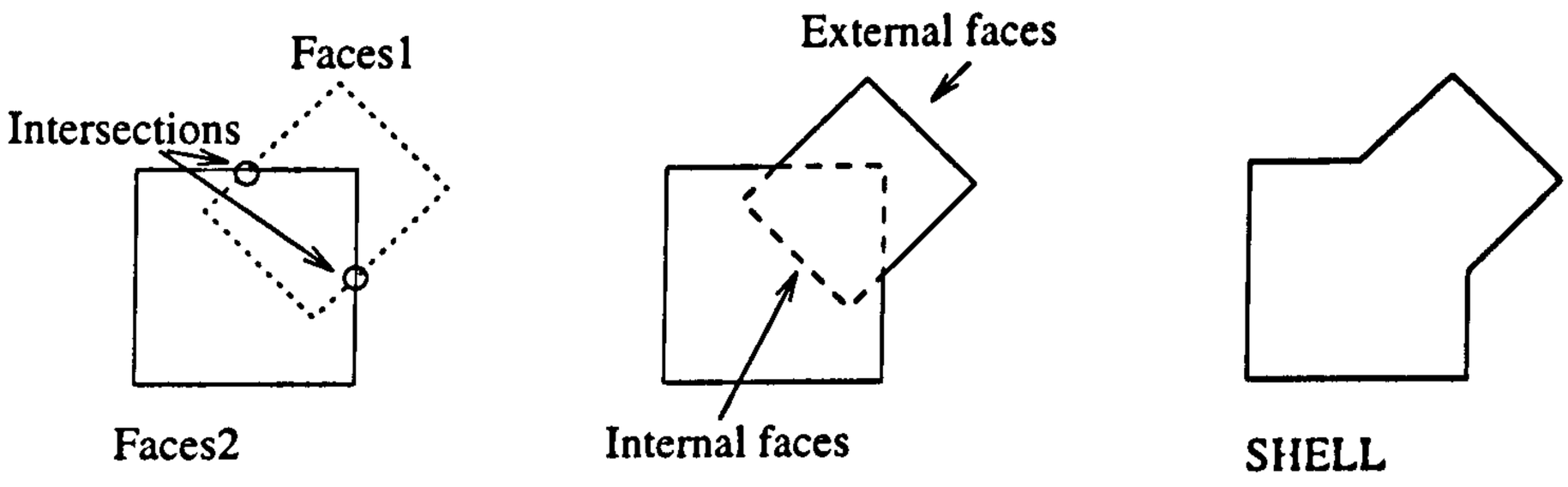
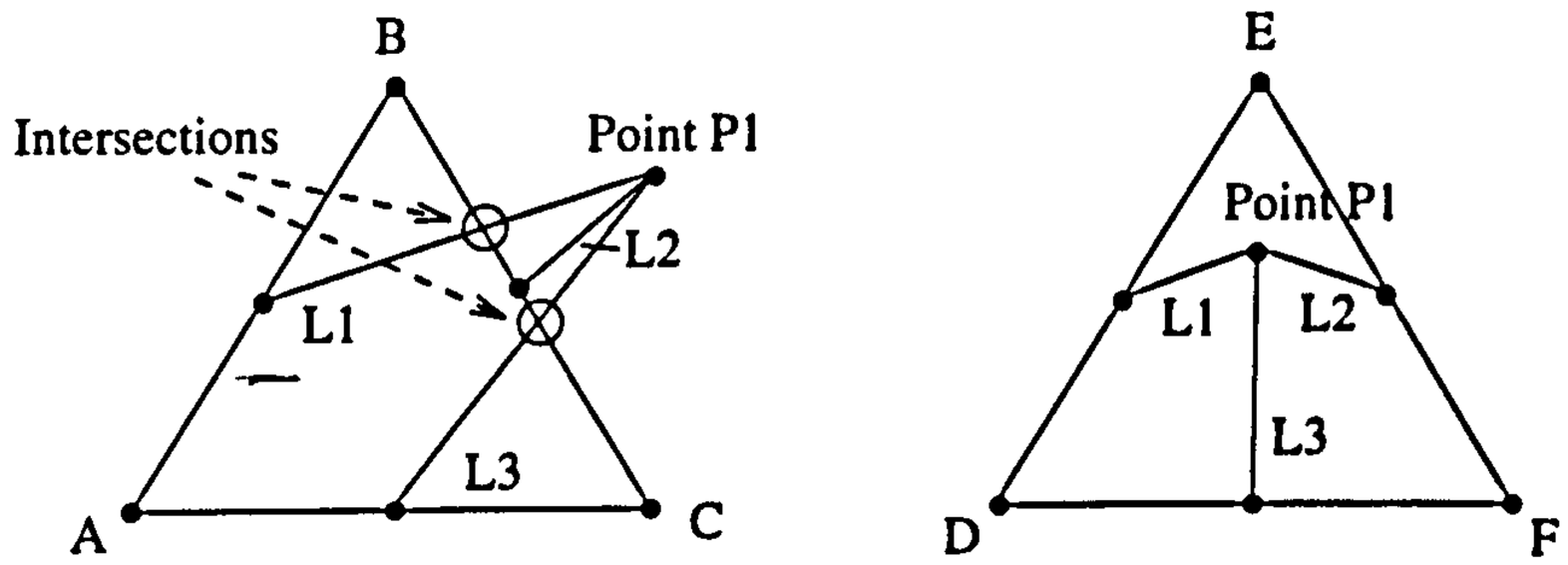


Figure 6.3 Constructing a shell from intersections in 2D

When this process is applied to 3D, each face in Faces1 must be tested against every face in Faces2. For each pair of faces tested, the intersection will occur where the edges of one face intersect the other face. Calculating the intersection of two faces is done here in two steps. First the intersection of the edge of one face with the plane of the other face is found, if indeed it exists at all. Since the vertices of a triangular face define a plane, the edge-plane intersection is mathematically the intersection of a line segment and a plane, as described in Appendix A.1.

If this edge-plane intersection occurs, it must then be determined whether the point of intersection lies within the triangular face itself. To do this, the triangle and the intersection point are transformed into a 2D coordinate system in the plane of the triangle, as described in Appendix A.2. In this 2D coordinate system, the transformed point, now called P1, is joined to the midpoint of the triangle's sides to create three lines which are labelled L1, L2 and L3 in Figure 6.4.



a) Point P1 lies outside ABC

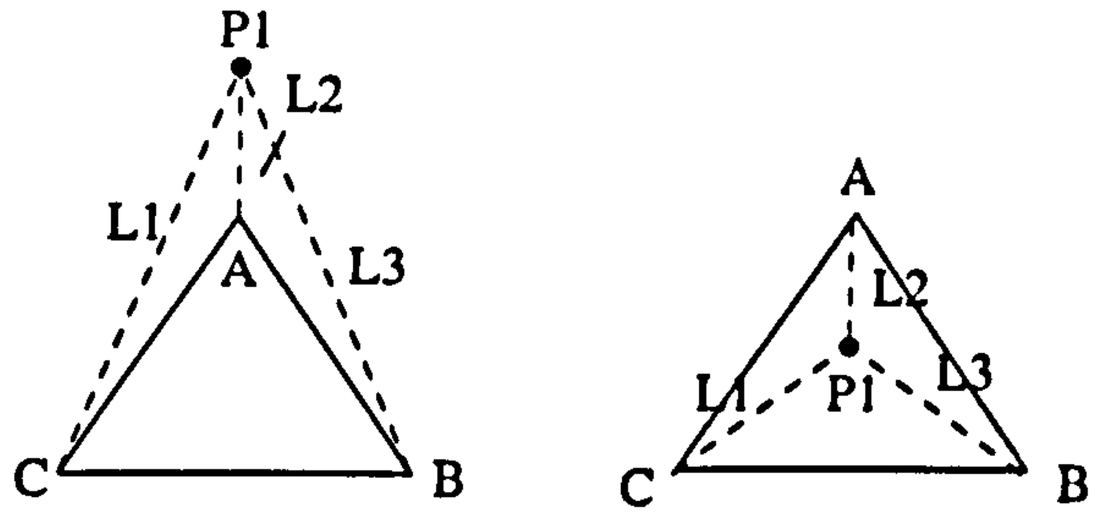
b) Point P1 lies inside DEF

Figure 6.4 Point lying outside or inside a triangle

Intersections between lines L1, L2 and L3 and the sides of the triangle can be found mathematically as the intersection of two line segments, described in Appendix A.3. As shown in Figure 6.4a, if any line-side intersections are found in 2D, then P1 lies outside the triangle, and hence the intersection point of the edge and the face in 3D also lies outside that face. If no line-side intersections are found in 2D, however, then the intersection point lies within the face, as is shown in Figure 6.4b.

In this program, the lines L1, L2 and L3 were joined to the midpoints of the triangle edges for convenience in calculations. But it is important that the endpoints of L1, L2 and L3 lie on the triangle's edges rather than their apexes. This is because this gives a unique set of results in the line-side intersection test. Figure 6.5 illustrates what would happen if the apexes of the triangle were used instead. Here there are no line-side intersections both when the point lies

inside the triangle ABC and when the point lies outside the triangle.

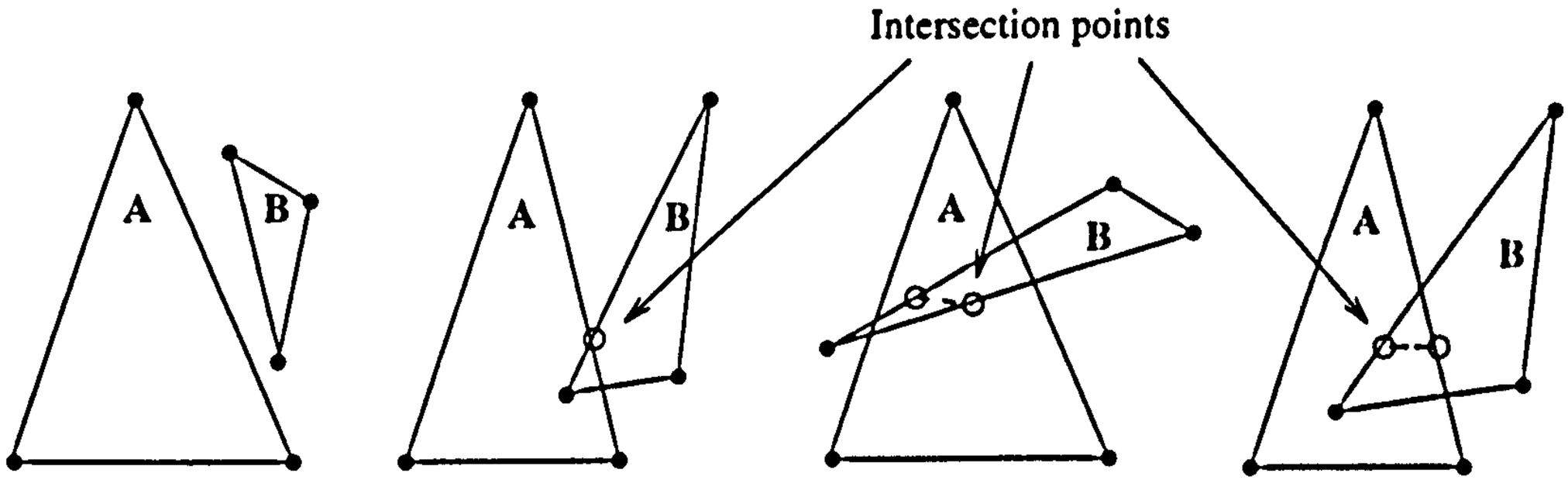


P1 lies outside triangle
No line-side intersections

P1 lies inside triangle
No line-side intersections

Figure 6.5 Lines to apexes give non-unique solution

So far, we have shown how to find the intersection point between one edge of a face and another face. In order to find the complete intersection between two triangular faces, each edge of one triangle must be tested against the other triangle. As can be seen in Figure 6.6, for any two triangles there can be zero, one or two intersections.



a) No intersection

b) Intersecting edges

c) Intersection points lie within triangle A

d) Intersection points lie on edges of triangles A and B

Figure 6.6 Intersecting triangles

After the edges of one triangle, triangle A, have been tested against the face of the other

triangle, B, the triangles may need to be swapped around and the edges of triangle B tested against the face of triangle A. This will ensure that all possible intersection have been found. The occasions when such a swap is necessary are presented in Table 6.1.

No. of intersections found	Status	Further action
2	Complete face intersection: Figure 6.5c	None
1	The faces straddle each other: Figure 6.5d	Swap triangles A and B so that the edges of triangle A are tested against the plane of triangle B
1	Edges intersect: Figure 6.5b	None
0	The edges do not intersect the faces of triangle A: Figure 6.5a	Swap triangles A and B as above.

Table 6.1 Options when intersections have been found

Once the face-face intersection points have been found, these faces must be subdivided into smaller triangular faces. This subdivision depends on the position of the intersection points for each triangle, that is, whether the intersection points lie inside the triangle, on an edge, or at an apex, as follows;

1. An intersection point at the apex of a triangle does not divide that triangle.

2. An intersection point at an edge divides the triangle into two triangles.

3. Where two intersection points lie inside a triangle, the first point inserted divides the

triangle into three smaller triangles by creating edges between the point and the triangle apexes. The second intersection point is then inserted into the triangles using a 2D Delaunay mesh generation method which has been outlined in Section 5.2.

Figure 6.7 shows some of subdivisions that may be made. Where a choice occurs between subdividing a quadrilateral along one of its two diagonals, the shorter diagonal is always chosen, which creates the best quality triangles.

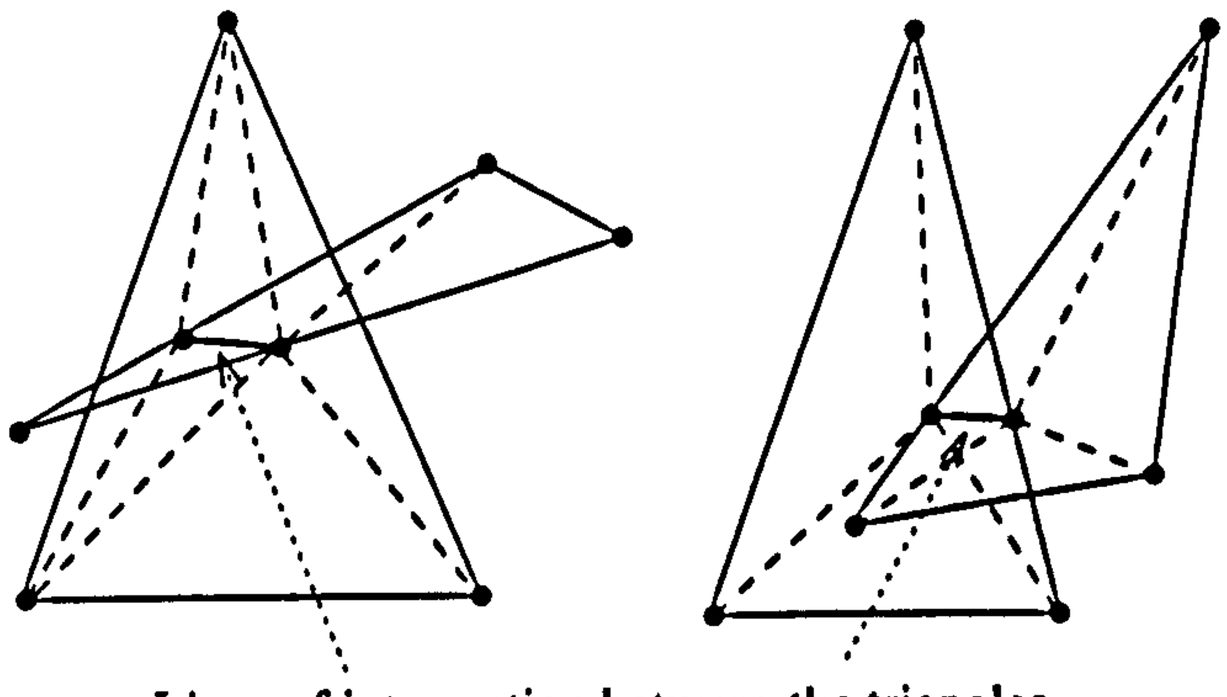


Figure 6.7 Subdividing intersecting triangles

When all of the intersecting faces in Faces1 and Faces2 have been subdivided, the outer shell of faces is listed, as was illustrated in 2D in Figure 6.3. Since Faces1 and Faces2 overlap one another, there are some faces in Faces1 that lie inside the shell of Faces2 and vice versa. A test for vertices lying inside a shell of faces is described in Appendix A.4. This test is used to determine which faces lie on the outside of the combined set of faces. These are listed in SHELL, as shown in Figures 6.1 and 6.3. SHELL is therefore a closed set of triangular faces which bound the void between the two initial meshes, MESH1 and MESH2.

The SHELL faces which contain the vertices created at the intersection of the two MESHes

will lie on the surface of the Final mesh and do not adjoin any existing cells in the REST or Mesh cell sets. As can be seen from Figure 6.8, the surface faces are usually of poor quality due to the inclusion of the intersection vertices.

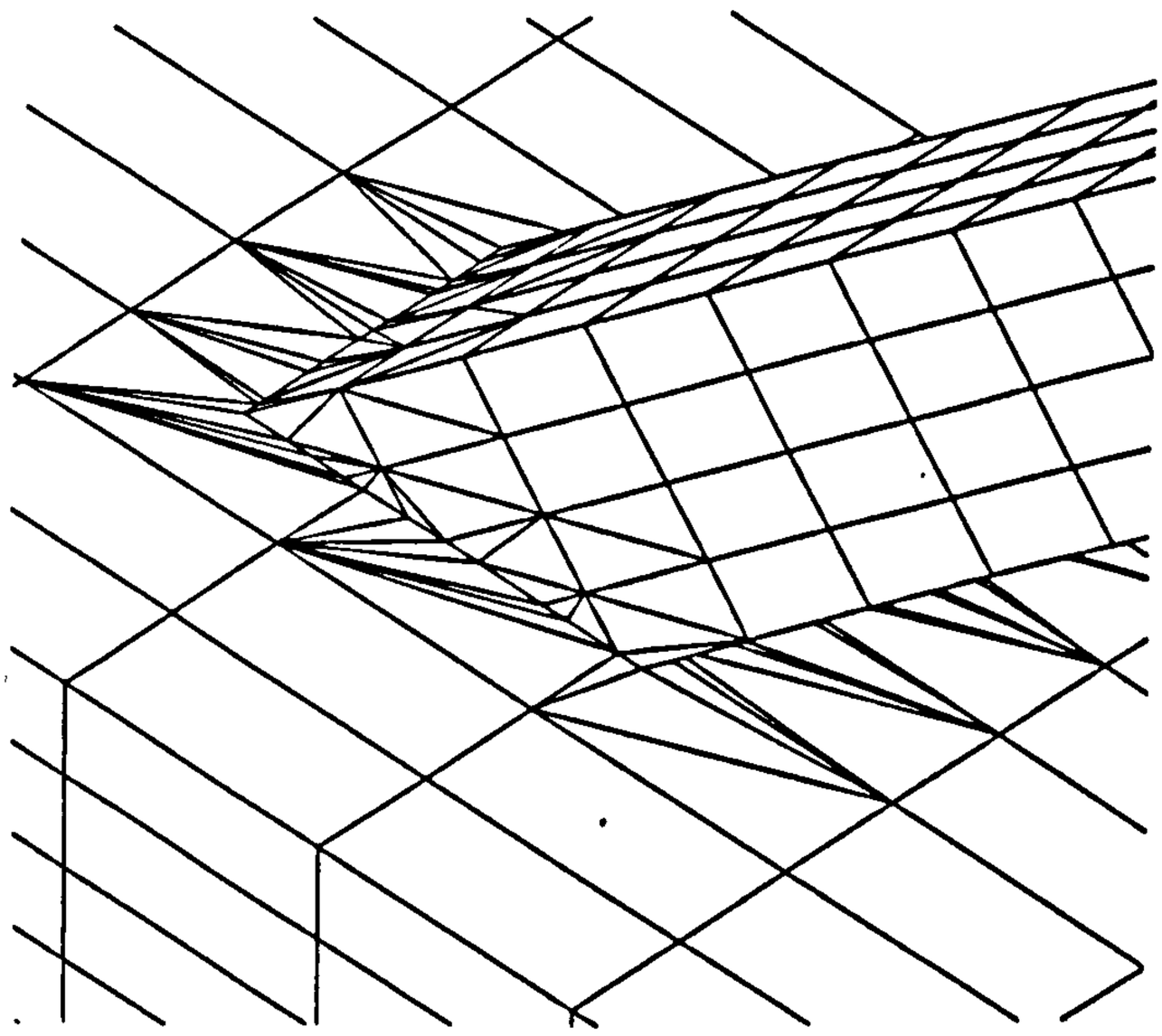


Figure 6.8 Large numbers of poor quality cells at the intersection

These poor quality surface faces will produce poor quality tetrahedral cells within SHELL and so the faces should be refined before the tetrahedral mesh is generated. The surface refinement can be done using 2D edge-swapping and vertex-merging techniques which are described below.

The edge-swapping routine tests each edge of the triangular faces in SHELL that have been subdivided. Every edge in this set of faces splits two adjacent triangles along one diagonal of the quadrilateral that they define. A quadrilateral has two possible diagonals, and the best quality triangles are produced if the quadrilateral is split along its shortest diagonal. A check is therefore made to see if the existing edge is the shortest diagonal for the quadrilateral, if it is not, then the two triangular faces are deleted and replaced with the better quality faces defined by the shorter diagonal.

Since the triangular faces each lie in a plane, it is valid to treat them as 2D, but the quadrilateral defined by two adjacent triangles does not necessarily have all its endpoints in the same plane. Thus the above edge-swapping has only been performed when the angle between the two planes is less than 15^o. For greater angles, the surface of the mesh is altered too much. Angles greater than this are more likely to be caused by the geometry of the intersecting surfaces as opposed to approximations in the surface by the facets.

Vertex-merging is done where two points along the line of intersection lie too close to one another. These points are found by assessing the distance between two adjacent vertices which lie along the intersection. Where this distance is less than a user defined limit the two vertices are merged. The triangular faces which share this deleted edge are themselves deleted from SHELL. After surface refining, SHELL consists of a closed set of good quality triangular faces, ready for creating the tetrahedral mesh.

6.6 Cell creation within SHELL

A tetrahedral mesh is to be generated within SHELL without cell edges crossing the faces and without gaps or overlapping in the mesh. Two methods have been explored for this purpose, a Delaunay based tetrahedrisation method and the author's novel 'Meshstar' method.

6.6.1 Three-dimensional Delaunay method

The Delaunay method has been described previously in Section 5.2.4.2 for the 2D case. In 3D, the technique is essentially the same. A set of points which lie inside SHELL must be generated first of all. Ideally, the points will be distributed evenly throughout SHELL such that the best quality mesh will be created for that volume. It was noted, however, that a set of points already existed within SHELL which were those vertices listed in Delete1 and Delete2.

The two sets of vertices were written to a single point-insertion list which could then be used by the Delaunay method to generate a tetrahedral mesh throughout SHELL. A point

coincidence check was first made, and the tolerance for this check was the shortest edge length of the cells in Delete1 and Delete2. This ensured that the vertex spacing for the tetrahedral mesh was not smaller than that of the finer mesh and should therefore produce cells with a similar edge length to the original meshes.

One problem in the 3D case is that some tetrahedra created have zero volume or are long thin "slivers", which are discussed by Cavendish et al [75]. Zero volume tetrahedra occur when an inserted point lies on the same plane as a triangular face on the insertion polyhedron. This will cause discrepancies in the mesh and a CFD solution will not be possible. Slivers are produced in a similar fashion, if the inserted point lies too close to the plane of a triangular face. This creates a tetrahedron with a very large circumsphere, which will cause the Delaunay method to break down, as shown in 2D in Figure 6.9.

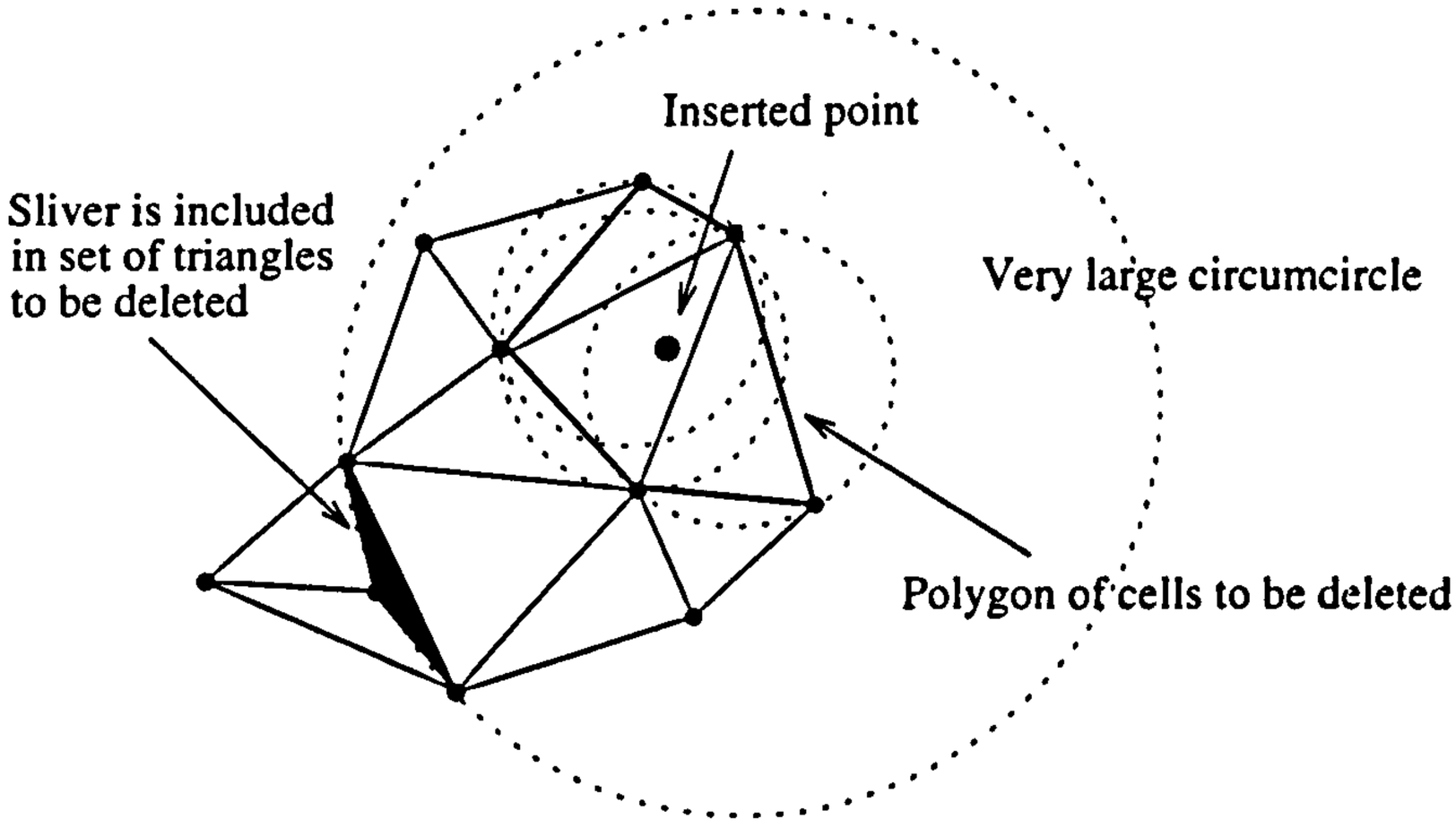


Figure 6.9 Breakdown of Delaunay method due to sliver

This Figure shows how a sliver can be included in the list of cells to be deleted, which will lead to two insertion polygons being created. When the inserted point is linked to the vertices of both polygons, the cells linked to the sliver will cross other cells in the mesh. Both zero-

volume tetrahedra and slivers can be avoided by checking the planes of the triangular faces within the insertion polyhedron. If the inserted point lies too close to this plane then it should be moved normal to the plane by a small distance. This should move the point far enough away from that plane so that slivers or zero-volume cells will not be generated. Since this coordinate adjustment may move the point outside of the insertion polyhedron originally calculated, a new insertion polyhedron must be found for the new point coordinates.

In the mesh construction procedure, the initial, large tetrahedron is defined using the limits of the working region. The cube defined by those limits are scaled up by 30%, which guarantees that all of the vertices within the working region lie inside a tetrahedron defined from the edges of that cube.

One aspect of Delaunay meshing is that the mesh generated will not necessarily match the boundary faces, as mentioned previously in Section 5.2.4.2. If the original cells in MESH1 and MESH2 are of good quality, then the faces of SHELL will generally match the triangles created from the vertices in SHELL by the Delaunay method. If the original cells are badly distorted the mis-match of faces cannot readily be corrected.

In order to make mesh construction robust for any type of original mesh another method of tetrahedral mesh generation was considered. In this work, the Meshstar method was developed to ensure that a coherent tetrahedral mesh which matched the faces in SHELL was created from the very start of the mesh generation.

6.6.2 Meshstar tetrahedral mesh generator

The Meshstar tetrahedral mesh generator starts off with the closed set of triangular faces, SHELL, and a new central vertex is created inside SHELL. Each face in SHELL is then linked up to this new central vertex, which creates a mesh of tetrahedral cells that match the boundary faces of SHELL exactly. Typically, these tetrahedral cells are very distorted and so a two-stage mesh refining process of 3D edge-swapping and vertex-insertion is used until an

acceptable level of cell distortion is reached. These three processes, central vertex definition, edge-swapping and vertex-insertion are outlined in the following subsections.

6.6.2.1 Central vertex placement

The central vertex is defined as the average coordinate position of all the vertices in SHELL. The tetrahedral mesh will then be generated inside SHELL by construction each triangular face up to the central vertex. The central vertex can be badly placed, however, where the edges of the tetrahedral cells cross the SHELL faces. This face-crossing is tested for by finding the intersection of the plane of each SHELL face with the edges of each tetrahedral cell, as described in Section 6.5. If no faces are crossed, then the averaged vertex position can be used for the central vertex. If not, then the central vertex must be moved.

A variety of central vertex moving algorithms could be developed. For example, the surface face of the cell whose edge crosses other surface faces could be used to guide the movement of the central vertex. During the course of this work, however, it emerged that the search-distance influenced how many cells were deleted from the meshes, and hence the number of faces in SHELL. Crossed faces occurred where the surfaces of the overlapping meshes curved near the overlap, and if too many cells in this curving mesh were selected then crossed faces were inevitable. If the search-distance were reduced, however, fewer cells in the curving mesh would be selected, and so crossed faces were eliminated, as shown in Figure 6.10. In this figure, a larger search distance creates cell group 1, which produces cells crossing the SHELL faces. A smaller search-distance produces cell group 2, which can be seen to lie entirely with in SHELL faces.

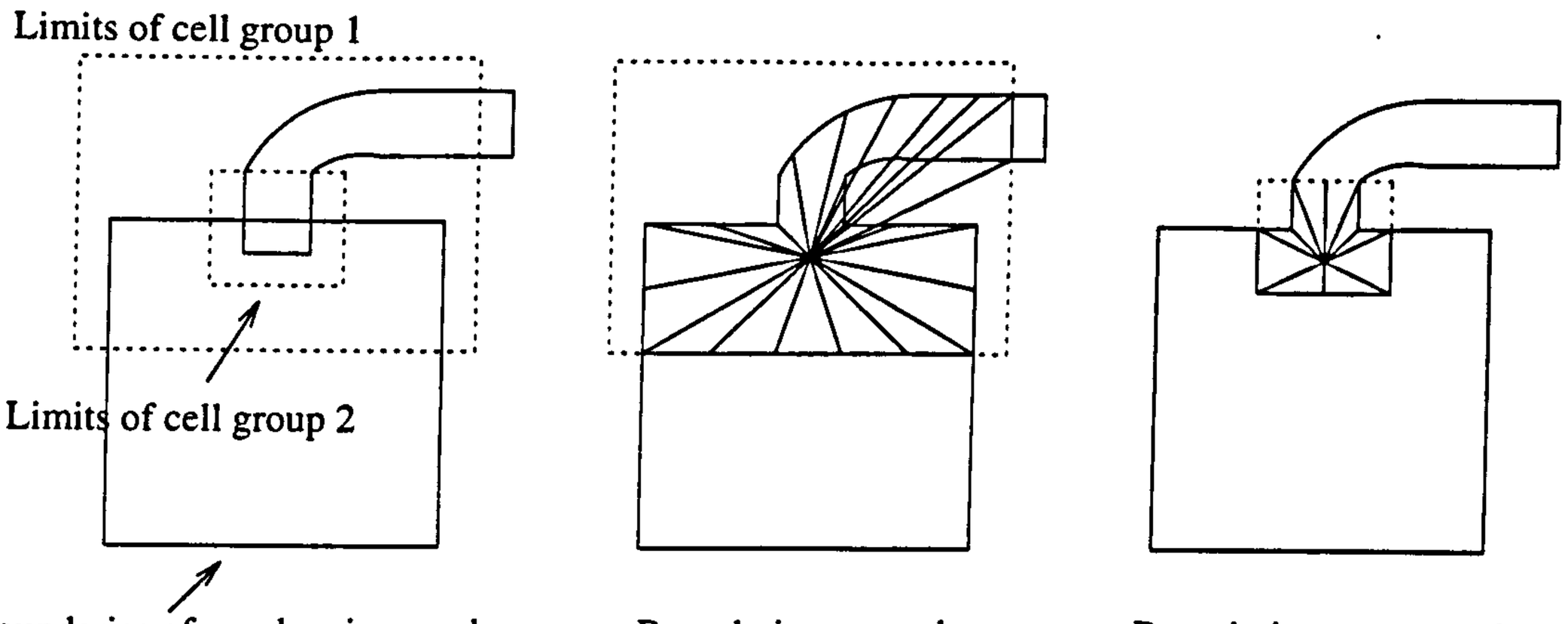


Figure 6.10 Search distance influencing tetrahedral mesh

This was a working solution to the problem and a deeper investigation of such crossed faces is recommended at a later date.

6.6.2.2 Edge-swapping

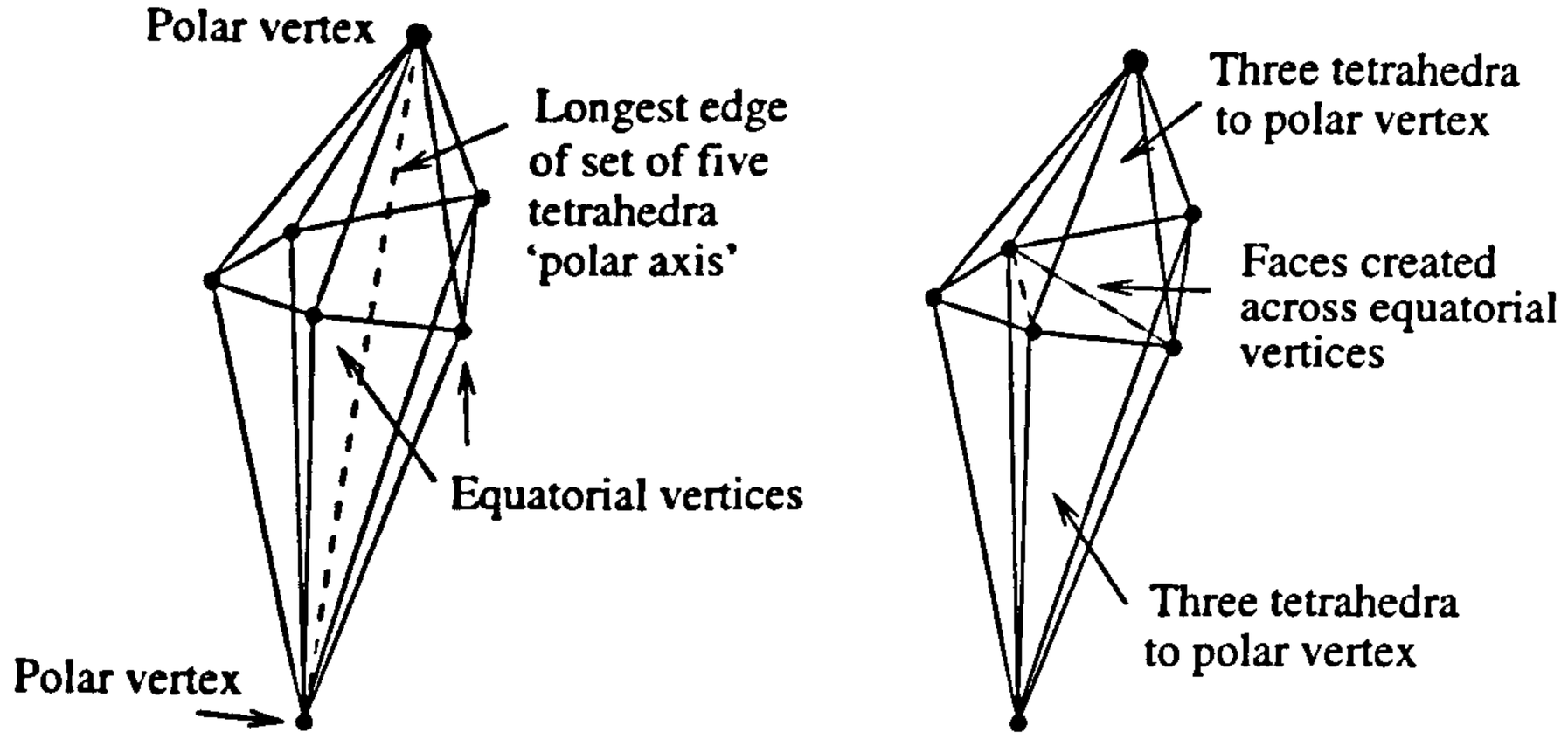
The swapping of edges described for 2D cases in Section 6.5 can be extended to the 3D tetrahedra created inside SHELL. In 3D, the best proportions for a tetrahedron are equilateral triangles on each face. An appropriate edge to be swapped is one that is the longest edge of all the tetrahedra that it is attached to. This longest edge must lie within the mesh and not on the surface of the mesh, because the surface of the mesh must not be changed during the tetrahedral mesh refining. Otherwise the surface faces will not be properly blended to the rest of the meshes in stage 4 of mesh construction. The edge lies within the mesh and can be swapped if the following relations are true;

number of vertices in tetrahedra set = n

number of edges in tetrahedra set = 3n + 1

If it is a valid edge for swapping, this longest edge forms a 'polar axis' through the set of tetrahedra. The vertices in the tetrahedra set that are not attached to the polar axis form an

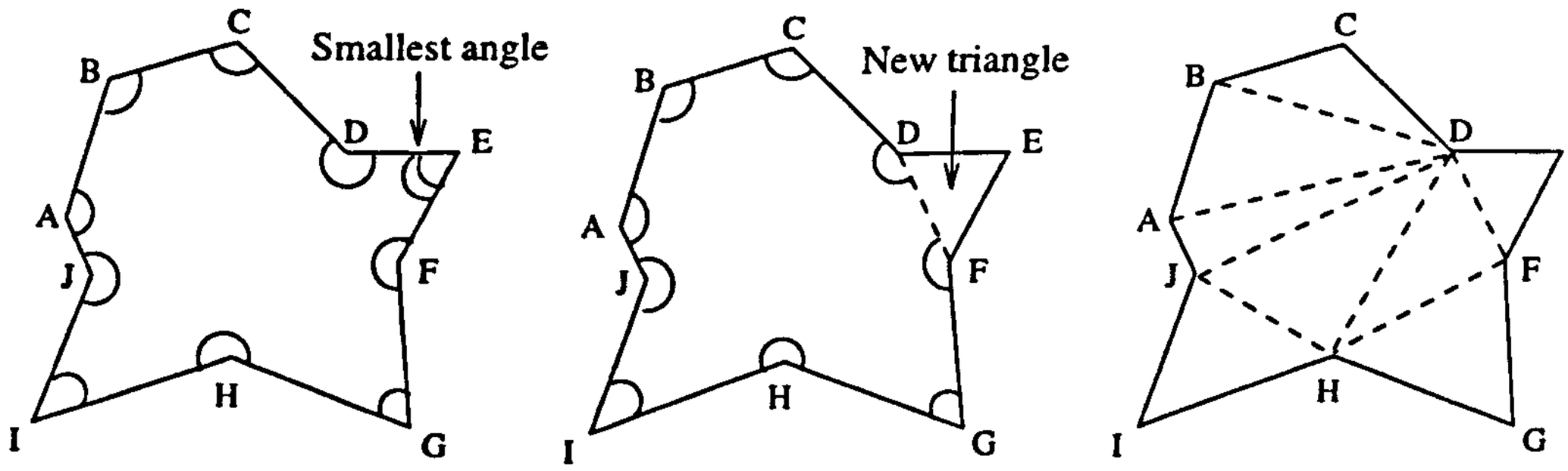
'equatorial' loop of edges as shown in Figure 6.11.



a) Longest edge is swappable b) Better quality tetrahedra

Figure 6.11 Edge-swapping in tetrahedra

The tetrahedra set is deleted from the mesh leaving a polyhedron of triangular edges surrounding a void. The vertices in this polyhedron are used to create a new set of tetrahedra to fill the void. First the edges between the equatorial vertices are considered, as is illustrated in Figure 6.12 below. It must be noted that although this Figure is shown in 2D, the actual set of vertices lie in 3D coordinate space, and not necessarily all in the same plane.



a) Find smallest angle b) Create edge subtending angle c) All edges created

Figure 6.12 Creating edges between equatorial vertices

First the angle between each pair of adjacent edges if found by projecting the edges into 2D on the plane that their three points define (see Appendix A.2) and by then using trigonometric cosine functions. The smallest angle is found, as shown in Figure 6.12a) where the angle at point E is smallest. For the smallest angle found, a new edge is created, thus forming a new triangle as shown in Figure 6.21b).

This new edge is added to the list of edges and the old edges of the triangle are deleted from the list. The process continues until a quadrilateral remains, and then the last edge created is the shortest diagonal of that quadrilateral. Figure 6.12c) shows the complete mesh fitted between a set of equatorial vertices. This technique is similar to the advancing front method, except that only the original boundary vertices are used to create the triangles, and no new vertices are created.

Once the equatorial vertices have a mesh of triangles fitted between them, these triangles are connected to the polar vertices, so forming a new set of tetrahedra. Where there were n tetrahedra in the original tetrahedra set, this method of 'edge-swapping' produces (2n - 4) new tetrahedra. Thus for tetrahedra sets of more than five cells, this method increases the number of cells as well as improving cell quality.

6.6.2.3 Vertex insertion

The mesh is also refined by inserting new vertices in areas of high cell distortion. The most distorted cell in the tetrahedral mesh is defined here as the cell containing the smallest internal angle in a face. Any cell attached to this deleted cell is itself deleted, leaving a void in the mesh surrounded by a set of triangular faces. A new vertex is placed at the centre of the volume bounded by the triangles in the position of the averaged coordinates of the vertices. A new set of tetrahedra are created by linking the vertex of this polyhedron to the new vertex. If any cell edges cross the faces of the polyhedron, the cell adjoining this polyhedron face is deleted, increasing the volume of the void, creating a larger polyhedron. This process continues until a coherent mesh can be formed from the inserted vertex.

6.7 Mesh assembly

The entire mesh is assembled from the following vertex and cell sets; REST, Mesh and TETRAS. The final mesh is written out with sequentially labelled vertices and cells without any gaps. The mesh can then be used for CFD analysis or it can be linked to another mesh to build up a mesh of a more complex geometry. In order to produce a single, coherent linked mesh, the SHELL faces must match those in mesh1 and mesh2. That is, a set of vertices are common to SHELL and mesh1 or mesh2, and the faces defined by these vertices must match. Any quadrilateral faces in the Faces sets were subdivided into triangles before they were included in SHELL. Mesh blending must therefore occur where the triangular faces in SHELL adjoin their parent quadrilateral faces in mesh1 and mesh2.

The first step in blending the faces is to find all the cells in mesh1 and mesh2 that are adjacent to the original cells in Delete1 or Delete2. The quadrilateral faces in these adjacent cells are divided into triangular faces and their parent cells are divided into tetrahedral and pentahedral cells. Indeed the cells used in mesh construction are solid linear hexahedra, pentahedra, wedges and tetrahedra. The following describes how different cell types may be subdivided along a quadrilateral face.

Pyramidal cell

The quadrilateral face of the pyramidal cell is split into two triangular faces. This will form two tetrahedra as shown in Figure 6.13.

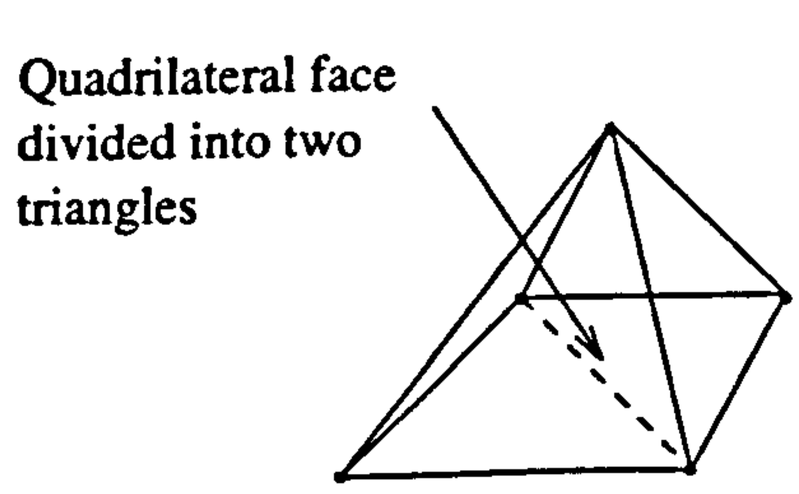


Figure 6.13 Subdivision of pentahedral (pyramidal) cell

Hexahedral cell

A new vertex can be created to the centre of a hexahedron. By joining each vertex in the hexahedron to this new vertex, five pyramidal cells and two tetrahedral cells will be created. Figure 6.14 shows a hexahedral cell divided at the central vertex.

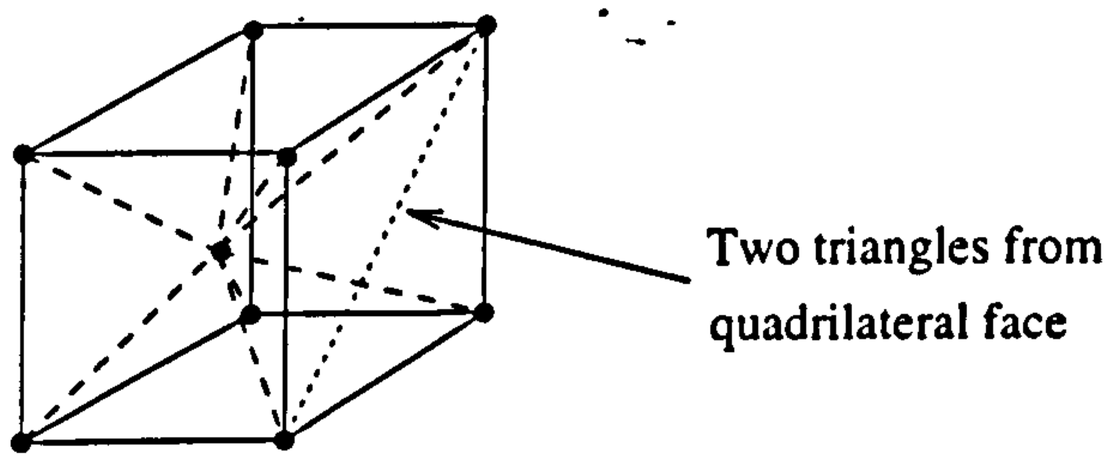


Figure 6.14 Subdivision of hexahedral cell

Wedge-shaped cell

The wedge cell has two triangular faces and three quadrilateral faces. If the triangular face is a boundary face, it requires no mesh blending. If a quadrilateral face lies on the boundary then it is divided into two triangles and a new vertex is placed at the centre of the cell. This creates two pyramids from the other two quadrilateral faces, two tetrahedra from the wedge's triangular faces and two tetrahedra from the split quadrilateral face. Figure 6.15 shows a wedge-shaped cell divided into tetrahedra and pyramids in this way.

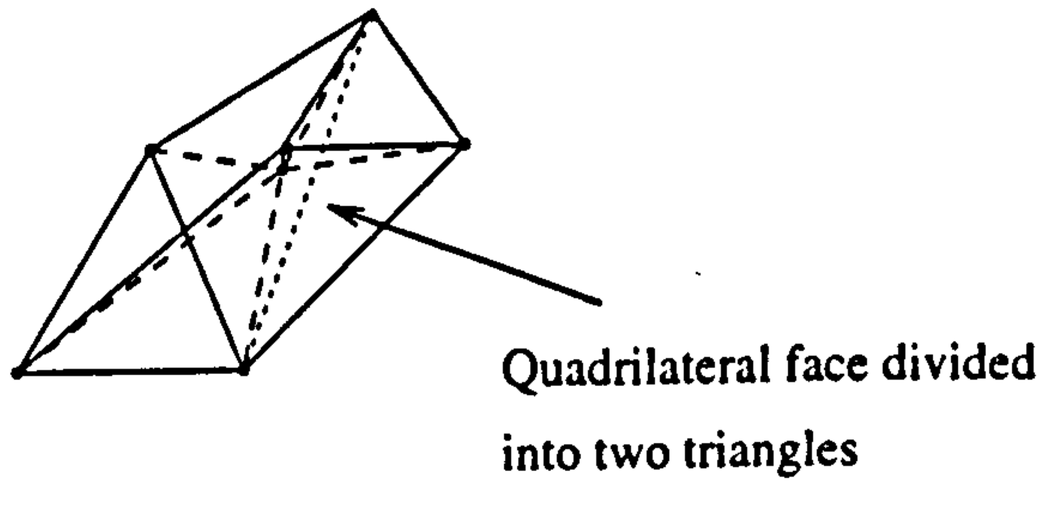


Figure 6.15 Subdivision of wedge-shaped cell

6.8 Mesh cutting

Similar techniques to those used for mesh linking are used when one mesh is to be cut from another. Once again any type of cell can be used in either mesh. In addition, the mesh that cuts the other may be a shell mesh instead of a 3D mesh of solid cells. This gives the user the freedom to use any cell type that is appropriate.

The four stages of mesh construction described in Section 6.3 are used, except that an extra operation is added, which is illustrated in Figure 6.16.

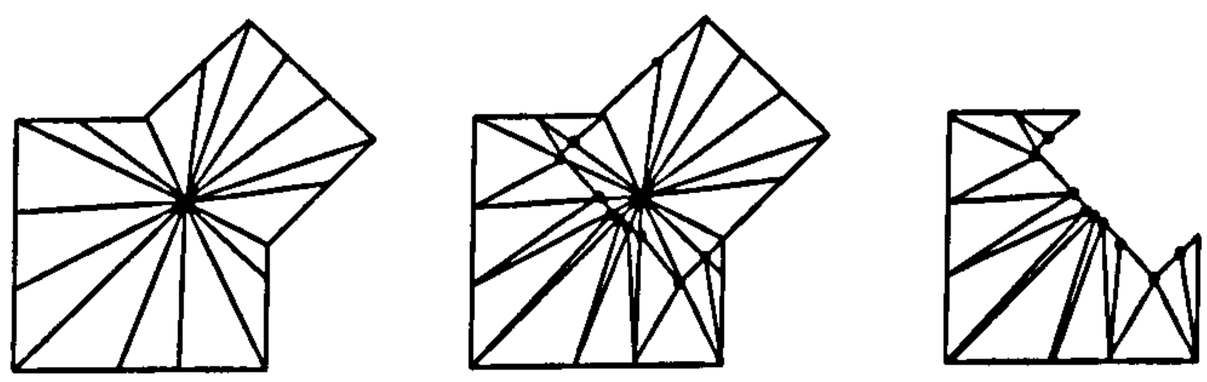


Figure 6.16 Mesh cutting process

After a mesh has been created within SHELL, stage 3 of mesh construction, the surface of the cutting mesh is then used to split the tetrahedra along that surface. First the edges of the tetrahedra are tested against the faces of the cutting shell. If an intersection point is found, then the tetrahedra that contain this edge are all subdivided by the intersection point. Next the edges of the cutting shell are tested against the faces of the now-modified tetrahedral mesh. Again the tetrahedra are subdivided where an intersection point is found. This defines a new surface within the tetrahedral mesh. Any cells which lie inside the cutting mesh are then deleted. The surface of the tetra mesh is now refined to remove highly distorted triangles and the tetrahedra are refined as before to give better quality

cells. The assembly of the tetrahedral mesh into the original mesh then proceeds in stage 4 as before, with the exception that no cells in the cutting mesh are included in the final mesh.

6.9 Mesh construction in context

Whilst the mesh construction program is unique, there are two programs with some aspects in common with mesh construction that have been developed in parallel.

The first is a mesh fitting code developed at the Aircraft Research Association (ARA) [76] based upon a 2D pilot study by Weatherill [77]. This code is concerned solely with external flow situations. In these cases a mesh ranges from the surface of the body to a distant boundary. In particular, the ARA code has been developed for aircraft studies. Topologically, an aircraft body and its wings do not form a complex geometry and can be meshed readily using basic mesh generation techniques. Additional features to the aircraft such as missiles, engine nacelles and under-carriages pose greater problems in mesh generation, however. ARA's code therefore takes a simple mesh of the aircraft body and wings and remeshes locally around the additional features required.

The technique is to define a block of cells in the regions of the feature and delete them. Pentahedral pyramids are then created inside the resultant gap in the mesh on the newly exposed quadrilateral surfaces. Points are then placed with the region of the deleted cells and on the surface of the feature that will be added. Delaunay methods are then used to generate a tetrahedral mesh fitted through these points.

This is very similar to the original design of the mesh construction program for joining two meshes, except that mesh construction also calculates the intersection of the surfaces of the meshes. Surface intersection is unnecessary for the external flow meshes used for aircraft and so it is not an aspect of the ARA code.

In addition, the original external flow mesh used by ARA is generally a mesh of good quality, hexahedral cells. The mesh construction code does not assume good quality cells in the initial mesh, and indeed when poor quality cells were used for the original meshes, the Delaunay mesh generation failed, as described in Section 6.6.1. This is why the mesh construction program now uses an initial tetrahedral mesh to fill shells which is then refined, as was outlined in section 6.6.2.

A second product similar to mesh construction is the mesh generator for the commercial CFD code PAM-FLUID [78]. In this code, a mesh of hexahedral cells may be cut by a shell of quadrilateral cells. The technique finds all the hexahedral cells that lie inside the shell and those cells that cross its boundary. These hexahedra are deleted and the vertices on the newly exposed cell faces are projected onto the surfaces defined by the shell. Local smoothing is then performed to improve the quality of the now-distorted cells.In using this product, the user should be careful to give the original mesh a high cell density in the cutting region so that the surface can be well defined and so that the hexahedra suffer as little distortion as possible.

Although these two products have some aspects in common with the author's mesh construction program, neither can be used for both joining and cutting meshes, including surface intersections. The mesh construction program offers a very flexible approach to creating meshes which could significantly reduce the time and effort required to generate a mesh for a complex geometry. It is not intended to be a sole mesh generator, however, rather it will complement other mesh generators making them more effective tools for complex geometries. Indeed, for simple geometries the user may well find more basic mesh generators to be the most appropriate tools.

The skill required to mesh a complex geometry is also reduced using mesh construction since the program requires very little user input. In particular, mesh density in the region of the mesh overlap is automatically set to be slightly finer than the original meshes used. Indeed, when considering the physics of fluid flow, changes in the flow variables

are significant where the geometry of the flow regime changes. For CFD analysis, this is the very region that should have smaller cells in order to capture the flow changes.

The mesh construction program uses tetrahedral cells and unless the original meshes had only tetrahedral cells, then the resultant mesh will have a mixture of cell types. STAR-CD supports the range of mixed cell types used in the program, but it may produce poor results if collapsed cells, that is, non-hexahedral cells, are used in regions with high flow gradients. Unfortunately this is precisely the region where mesh construction places tetrahedral cells and so the program may not be used confidently in conjunction with STAR-CD.

ARA encountered the same problem with flow solvers and so is developing its own CFD solver called SAUNA [79] which uses a mixture of cell types in any region of the flow. Another commercial CFD code which can use mixed cell types is Fluent-Unstructured [80]. This code was developed using formulations from the RAMPANT CFD code, which uses all tetrahedral cells. There is therefore no problem applying tetrahedral cells at changes in the geometry of the flow regime with this code. Indeed the developers of Fluent-Unstructured believe that this mixing of cell types will be a common trend in future commercial CFD codes, simply because it does allow a greater freedom in mesh generation [81].

6.10 Summary of chapter 6

A novel method for constructing complex meshes from meshes of simpler geometries has been presented. This is a 3D linking or cutting of any two meshes that are formed into a single coherent mesh using tetrahedral cells. The method requires the user to do the easiest forms of mesh generation on simple geometries using the most convenient mesh generation techniques and cell types. The program then automatically performs the difficult part of the process which is creating a single mesh of the complex geometry.

7. MESH CONSTRUCTION: APPLICATION

7.1 Introduction

The ideas and techniques for mesh construction outlined in chapter 6 were written into a computer program by the author. In this chapter, the use of the program is described and the program's application to some test cases is presented. Section 7.2 outlines the scope of the program and section 7.3 describes the data structures used. User inputs to the program are discussed in section 7.4 and section 7.5 describes the output from the program. Four test cases are then described in detail in section 7.6. Possible future developments and enhancements to the program are discussed in section 7.7 and a general discussion of the program is given in section 7.8.

7.2 Program scope

The mesh construction program was designed to make the linking or cutting of two existing meshes a simple operation. The user should have as little input as possible and restrictions on how the original meshes are generated should be minimised. If these aims are achieved then the program will be flexible and useful for generating meshes in a variety of flow situations.

In order for the user to use any technique or tool to generate the initial meshes, the program was designed with its own internal data structure for the meshes. The only information required from the initial meshes is vertex labels and positions in a 3D Cartesian coordinate system together with cell labels and connectivities. This data can generally be written out from the original mesh generators as ASCII files. The format of these data files is programmed into a file translation subroutine written by the author and this is run at the beginning of the mesh construction program. Currently, the translator is set to read data in Universal file format, whether written by I-DEAS or STAR-CD. In addition, the user may use any cell type in the original meshes chosen from a set of types listed in Table 7.1.

Cell type	Dimension	Vertex numbering
Triangle	2D	1 2 3 3 0 0 0 0
Quadrilateral	2D	1 2 3 4 0 0 0 0
Hexahedron	3D	1 2 3 4 5 6 7 8
Tetrahedron	3D	1 2 3 3 4 4 4 4
Pyramidal pentahedron	3D	1 2 3 4 5 5 5 5
Prismatic wedge	3D	1 2 3 3 4 5 6 6

Table 7.1 Cell types and numbering

Indeed the initial meshes used in mesh construction can also contain a mixture of cell types. As was seen in chapter 6, the mesh construction program uses the cell face data and so any other cell type may be included into the program easily provided that the cell faces are either triangular or quadrilateral.

Further benefits to the user would be achieved if the program is both easy and fast to use. In the design of the program, the user has very little input into the program, making it very easy to use, and these inputs will be discussed in the next section. However, computing speed and algorithmic efficiency have not been addressed in this version of the program. Improvements in speed and efficiency are likely to be the most important developments of the program in the future.

Mesh construction uses SI units throughout, and all vertex coordinates read into the program must be in SI units. To make the program more flexible a subroutine could be written which converted any units to SI, which would be an addition useful for industrial use of the program. The other assumption made in the program is that vertex and cell labels are sequential and start at one. This is a simple thing to ensure when using I-DEAS or STAR-CD, but again a conversion subroutine could be written which would make the mesh construction program more robust for use in industry. Hence even fewer restrictions would be placed upon the

generation of the original meshes.

7.3 Data structure

The mesh construction program operates upon sets of vertices and cells. The vertex coordinates are stored as a set of three real numbers in an array. Thus NODE(X,3) defines an array with X vertices, each with three real coordinates. The vertex label is also X so that a vertex is always called and referred to by its label. Where sets of vertices are grouped, they are stored as lists of vertex labels.

The cell connectivities are stored in an array CELLS(X,8) for all cells. Again the cell label, X, is also its position in the CELLS array and so the cell connectivity is called directly by the cell label. The eight array spaces for each cell are filled by the vertex labels for each vertex that defines the cell. Whilst a hexahedral cell contains eight vertices, other cells such as pentahedra and tetrahedra do not. Non-hexahedral cells are listed so that the cell type is evident by filling the excess array spaces with zeroes, as was shown in Table 7.1.

In remeshing, the cell labels and connectivities are continually changing as cells are replaced and new vertices created. In order to find those cells which need refining, the relation between cells and edges and vertices and edges needs to be determined. For example, in edge-swapping, the program needs to find all the tetrahedra which share a given edge. One method is to search all the tetrahedral cells every time an edge is considered for swapping. Even on a moderately small mesh these continual searches would be very computing intensive and take a long time. As an alternative, remeshing has another data structure in addition to the cell labels and connectivities. For example, a tetrahedron is listed with not only the four vertices which define that cell, but also the edges and vertices associated with that tetrahedron, as illustrated in Figure 7.1.

Tetrahedra	V	V	V	V	E	E	E	E	E	E
Edges	V	V	No	T	T	T	T	.....	T	
Vertices	La	No	E	E	E	E	.....	E		

KEY:

V	Vertex label
E	Edge label
T	Tetrahedron label
No	Number of items that follow
La	Label of vertex/edge

Figure 7.1 Data structures for mesh construction

As is also shown in Figure 7.1, an edge is listed as the two vertices that define it and the labels of all tetrahedra which contain that edge. In remeshing, the number of tetrahedra associated with an edge varies as cells are deleted and replaced by refined cells. Thus the number of tetrahedra associated with an edge at any one time in the running of the program is listed in the third column of the edges array. This is then followed by the labels of each of these cells.

Similarly the vertex-edge data is stored as vertex labels, the number of edges which contain that vertex, followed by the labels of these edges. These lists of cells, edges and vertices are updated as each cell refinement occurs in remeshing.

In addition, the edges are listed with the lowest vertex label first so that searches for matching edges can be made efficiently.

7.3.1 Array size

From Figure 7.1, it can be seen that whilst the array defining the tetrahedra has a constant 10 columns, the arrays defining edges and vertices can have a large number of columns since a new column is added to the array as each item is found. This can cause some problems with the initial specification of array sizes in FORTRAN 77, the programming language in which

the mesh construction program was written. In this language, the size of all arrays must be specified before the program is compiled and executed. However, each intersection between two triangular faces creates two new vertices and up to seven new edges, each of which must be added to the appropriate arrays. Thus the array size required for a given application depends upon the number of triangular faces that intersect between the two meshes, and as a consequence the array size is highly problem dependent.

In order for the program to be able to cope with meshes with a large number of cells, these arrays must be set with a high number of columns initially, even though this is not required for many of the smaller cases run. The same is true for the number of rows in the arrays which define tetrahedra as well as edges and vertices. That is, the array size must be set to a very high number of rows initially so that the program can be used for meshes with a large number of cells.

The problem with using very large arrays for all applications is that when the program is compiled, the computer allocates memory for each array defined. Thus a large amount of memory is reserved for large arrays, even if only a small part of the array is actually used by the program. This is not an efficient way to use a computer since it ties up large amounts of memory and this may restrict the other processes that run on the computer. It can also mean that the program can be used only on computers with a large amount of memory, making the program less portable and therefore less useful generally.

This problem could be tackled in a number of ways. Firstly, the general array size is defined as a 'parameter' value at the beginning of the program. This means that a new value can be specified by the user editing one number in a program and it will be extended to all the appropriate array sizes. Effectively, a new 'size' of program is used for every run and resizing the program would be easy. However, it is not good practice for users to edit the source code in the everyday running of the program since a simple mistake can corrupt the code and stop it working. In addition, the user would have to know what array size to enter, which is not easily determined before the program is run.

A second solution to the array size problem would be to have a number of versions of the program installed on the computer at the same time. Each of, say, three versions would have a different array size parameter and they could be used for small, medium or large problems. But this approach also has some drawbacks. First of all the program consists of over 5000 lines of code. So three versions of the program along with the executable files which are created when the code is compiled can take up a lot of computer disc space even before the initial meshes and the solution meshes are stored. Once again this is not an efficient use of computing resources. In addition, the user would still need to know which version to use, and this will not always be an obvious choice.

A third solution is to rewrite the program in another language. Both FORTRAN 90 and C programming languages can use a 'variable array size' which means that the array sizes can be defined dynamically as the program is being run. This is a very effective use of computing resources since the arrays are automatically sized according to the particular application. However a complete rewrite of the program would be a significant task and has not been done in the course of this work.

7.4 User Input

The 'front end' of the mesh construction program is an interactive menu. The program requests certain information from the user and sets up the subroutines accordingly. The information is requested as follows:-

1. Enter the name of mesh 1.
2. Is this a shell mesh? (y or n)
3. Does mesh 1 cut mesh 2? (y or n)
4. Enter the name of mesh 2.
5. Enter the search distance for vertices.
6. Enter point coincidence tolerance.
7. Enter min, max of volume surrounding overlap region;

Enter co-ordinates X min, X max

Enter co-ordinates Y min, Y max

Enter co-ordinates Z min, Z max

8. Enter minimum acceptable internal cell angle.

Each of these eight inputs are discussed in the following subsections, although the first four inputs will be considered together.

7.4.1 Mesh labels input

The program first asks for the name of mesh 1. If the two original meshes are to be joined, it makes no difference which mesh is entered first. For mesh cutting, however, the program assumes that mesh 1 cuts mesh 2. This is important since the result of cutting mesh 2 from mesh 1 will be a completely different geometry, as illustrated in Figure 7.2.

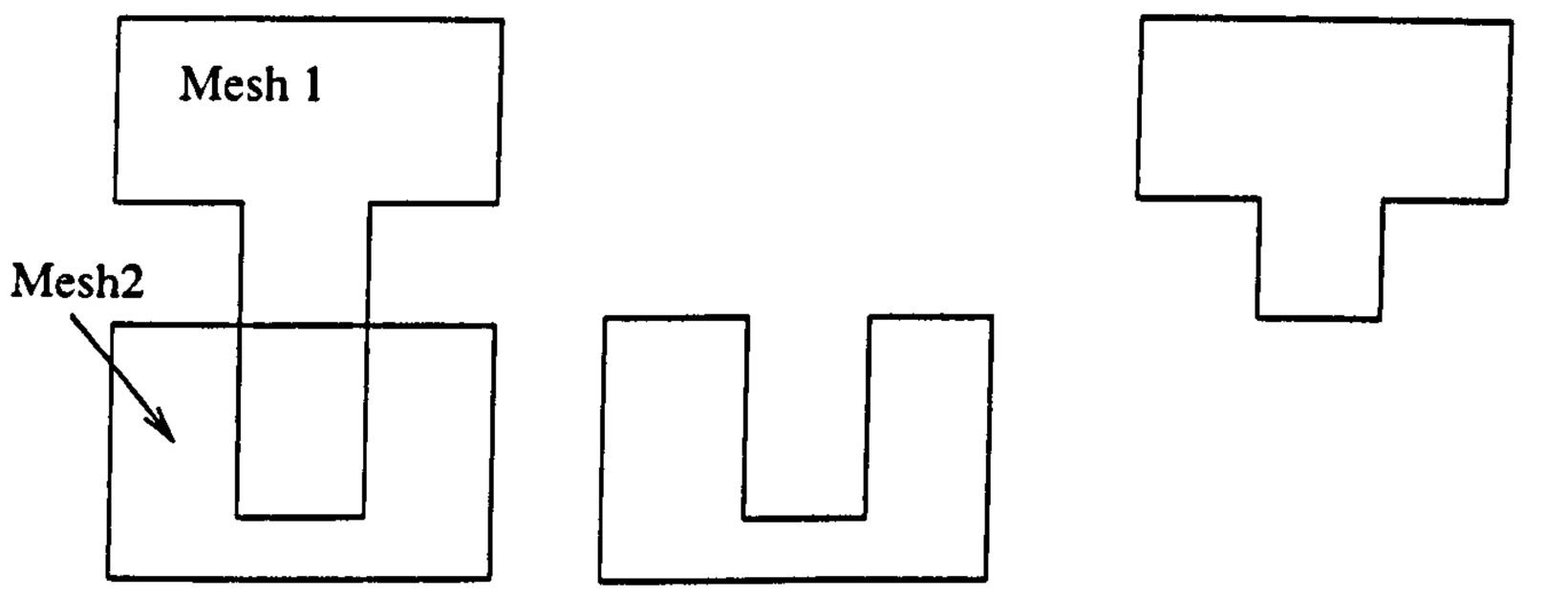


Figure 7.2 Effects of cutting mesh definition

For mesh cutting, the program enquires whether mesh 1 is a shell mesh. If it is not, then the surface shell cells are obtained as described in section 6.5. Next, the program requests the filename of mesh 2. For questions 1 and 4, if the filename entered is not found in the same

directory as the program, an error message is reported and the user is asked to asked re-enter the filename. Indeed, at any stage in the mesh construction program, the user may exit the program by pressing the "control" and "c" keys simultaneously. Whenever such a forced exit occurs the program stops immediately, with no recovery of intermediate stages.

Once again, it must be stressed that the two meshes should actually overlap one another rather than simply touch, for the reasons outlined in section 6.4. The mesh construction program was originally intended for operations on meshes with 3D cells, but mesh cutting can also use a cutting mesh consisting of shell cells. Due to the nature of the program, two meshes with shell cells can also be joined to create a single mesh either filled with 3D cells, or simply with shell cells. This makes it possible to create complex tetrahedral meshes from simple groups of shell cells, although this would probably not be the most effective use of the program.

7.4.2 Search distance input

The search distance was described in section 6.4.2 and is the distance between vertices in each mesh within which they can be considered to overlap. The cells found in this search will then be deleted from their respected meshes. The user can estimate the search distance by examining the cells in the overlap region visually in the mesh generation pre-processor, which for this work was either I-DEAS or STAR-CD.

As a general guide, the search distance should be a little greater than the largest cell side in the overlap regions. If the search distance is less than a cell side it is possible for a cell to be missed out in the search entirely. The program includes another 'layer' of cells extending away from the overlap region in order to smooth the concave 'gap' that remains when these cells are deleted. However, too small a search distance may lead to the deleted cells being too close to the actual overlap to allow good quality tetrahedral cells to be created. This is particularly true for mesh cutting where the surface of the cutting mesh extends into the overlap region.

If the search distance is large then a large number of cells are used in the calculations, which slows down the program. Also the curvature of the mesh surfaces near the overlap may be corrupted as described in section 6.6.2.1. The search distance technique is not the only method for determining which cells overlap, however. One alternative is to find the surface faces of each in turn and find those vertices from the other mesh that lie inside this shell. All cells associated with these vertices will be considered to lie in the overlap region. This technique is already used in other parts of the mesh construction program and could prove suitable. This would then further reduce the inputs required from the user and hence make the program more automatic.

7.4.3 Point coincidence tolerance input

The maximum distance between the two vertices within which they may be considered coincident is called is called the point coincidence tolerance. It is a very important feature of any mesh generator since computers may calculate extremely small distance between points that the user has intended to be coincident. This tolerance has been used freely within the mesh construction program for the following purposes;

Coincident points at face intersections

Points lying in a plane

Points lying inside or outside a shell

Point coincident with a line endpoint

Whilst the use of one global tolerance value has helped to minimise user input, it has also made the relation between each function very complex. Obviously the tolerance should be much less than the smallest cell side length in either mesh. Since the tolerance is also used to determine whether a point lies in a plane or not, much smaller tolerance values should be used.

7.4.4 Working region input

The extent of the working region is discussed in detail in section 6.4.1. The user must supply co-ordinates for a cube that surrounds the overlap region between the two initial meshes. These are entered as co-ordinate minima and maxima along each of the co-ordinate axes. The only constraint is that the working region is sufficiently large that it fully encloses the overlap region. If the working region is significantly larger than the overlap region, it will merely increase the number of cells considered by the program. This will not cause any errors, indeed the working region can completely enclose both meshes, but it defeats the object of having a working region in the first place if it is too large.

7.4.5 Internal angle input

The user may also select a minimum internal angle for the cells refined both on the surface intersection and within the tetrahedral mesh. If this angle is too small then poor quality cells may result. If the angle is large, however, the mesh created at the intersection may be so coarse the some of the surface detail is lost.

When running the program, if the user finds that this angle is inappropriate, then they can re-run the program with a different value. Indeed this iterative approach should be used for the search distance, point coincidence tolerance and minimum internal angle. The effects of each of these user inputs will be explored further for test cases described in section 7.6.

7.5 Program Output

Just as the program could read meshes described by various formats, so too can the final linked mesh written out using different formats for use with other software. Currently, the STAR-CD cell and vertex lists are written as input files 14 and 15 respectively. These can be read into PROSTAR using the CREAD and VREAD commands.

The mesh construction program also provides other details in its output report, which include the number of cells and vertices in the meshes. However cell quality data must be obtained by the user in the CFD software package used.

7.6 Applications

Four test cases are described and the sensitivity of the mesh construction program to input variables is explored. The program itself was divided into two computer programs in order to make programming faster and easier. The first program calculates the surface intersection of the two meshes and inserts the central vertex. Thus the mesh is complete with original cells, blending cells and initial tetrahedra. The second program finds the new surface if one mesh is cut from another, and also remeshes the internal tetrahedra to improve cell quality. Thus the first two test cases will illustrate the mesh linking in the first program, the third and fourth test cases will illustrate the operation of the second program which are mesh cutting and internal mesh refinement respectively.

7.6.1 Joining cylindrical hexahedral meshes

The first test case involves joining together two cylinders whose meshes have only hexahedral cells, as shown in Figure 7.3.

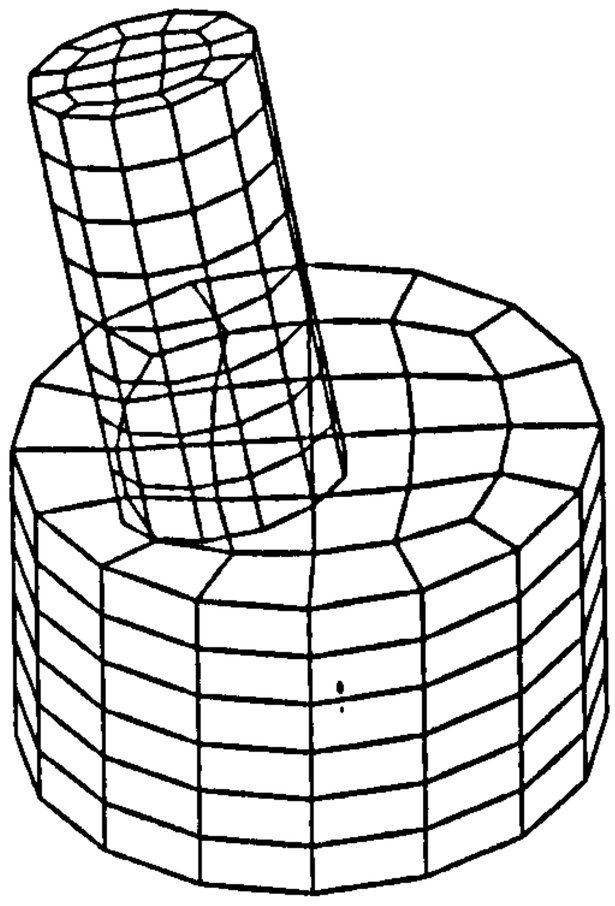


Figure 7.3 Overlapping initial meshes for test case 1

The narrower cylinder mesh has 192 cells and the wider cylinder mesh has 168 cells. A typical cell side length for both meshes was 7.5E-02m. The actual units of measurement used are irrelevant for the mesh construction process, and so from here SI units will be assumed to apply throughout.

A number of mesh construction runs were performed using these meshes and different user inputs. The first user input explored was the point-coincidence tolerance, ranging from 1.0E-12 to 1.0E-03. The results of this are outlined in Table 7.2 below.

Tolerance	Result
1.0E-12	Error message, program stopped
1.0E-09	Program ran OK
1.0E-06	Program ran OK
1.0E-05	Program ran for 2 days, stopped by user
1.0E-03	Program ran for 2 days, stopped by user

Table 7.2 Point coincidence tolerance for test case 1

From Table 7.2 it is clear that a point coincidence tolerance ranging between 1.0E-09 and 1.0E-06 allows good results to be obtained for the mesh construction program. Tolerances of 1.0E-03 and 1.0E-05 are about 1% of the average cell side length and were found to be too large. For these tolerances, the mesh construction program ran for over 48 hours without finishing and had to be stopped by the author.

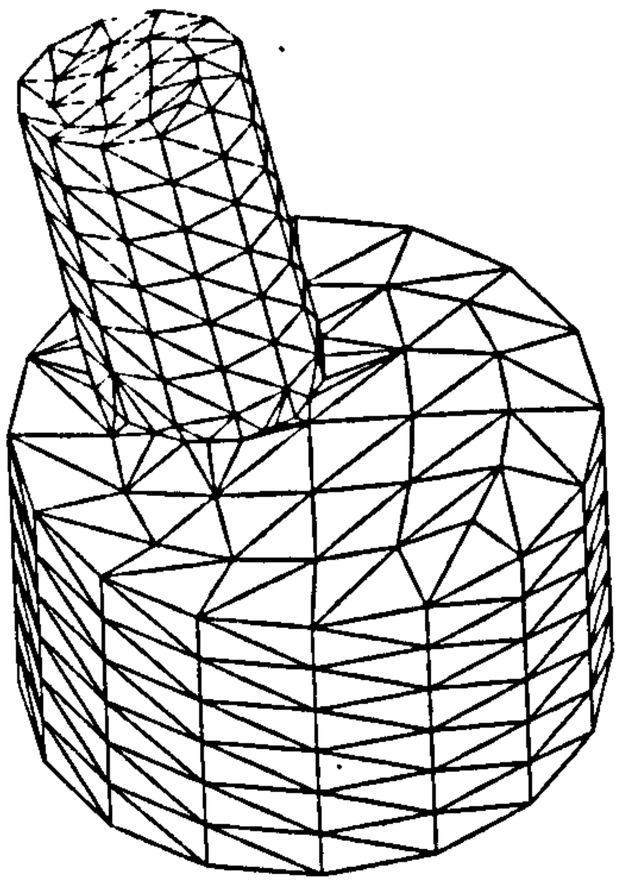
When the tolerance is too large, problems occur when the program determines whether points lie in a plane or not. Points not actually lying in the plane can be classed by the program as lying in the plane when the tolerance is too large. Thus incorrect intersections between faces may occur which then significantly increases the number of new faces created. Eventually the

array space allocated for the new faces is exceeded and the program overwrites array space and so continually searches the new and old faces without ending. Checks can be programmed so that the run is automatically stopped when an array size is exceeded. These checks should be included in future versions of this program since this would make the program easier to use.

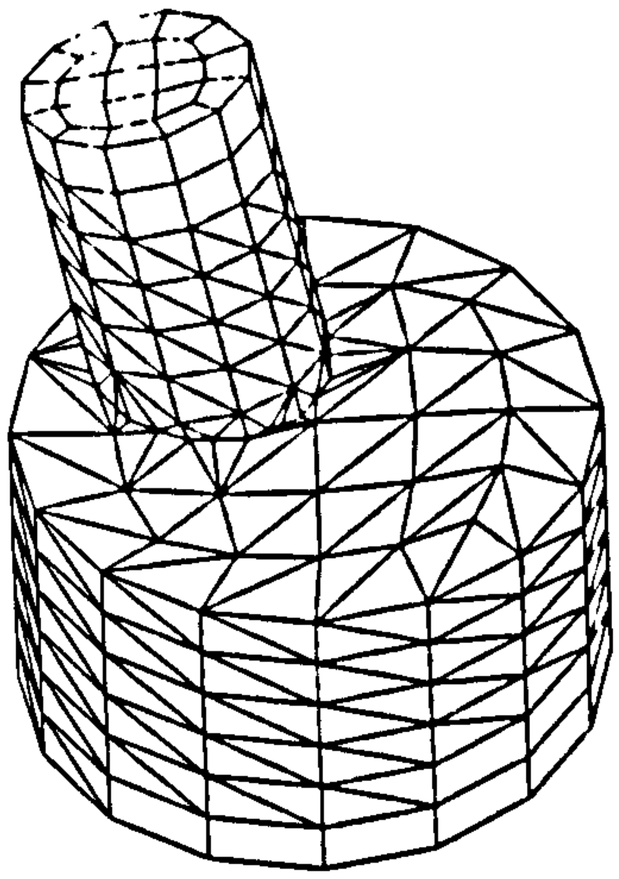
A tolerance of 1.0E-12, about 1.0E-09% of the average cell side length, was found to be too small. In this case, the program stopped with an error message "Division by zero" where what should have been coincident points were considered to be separate points by the program with very small distances between them.

Using a constant tolerance of 1.0E-09, the influence of the search distance was explored next. Since the meshes in this test case were small, the working region was set to include both meshes entirely so that the effects of varying the search distance could be seen clearly. Figure 7.4 overleaf shows plots for a range of search distances from 2.8E-01 to 0.1E-02.

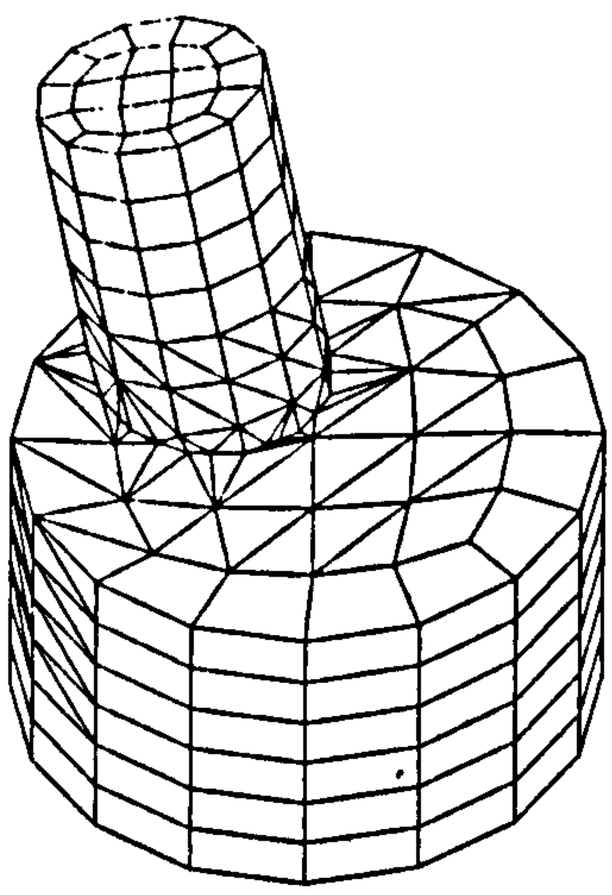
The search distances 2.8E-01 and 2.0E-01 both encompass the all of the meshes used. Thus all the hexahedra are converted to tetrahedra as is shown in Figure 7.4a. Search distances of 1.0E-01 and 7.0E-02 give a suitable region for the mesh linking with successively smaller numbers of cells considered as shown in Figures 7.4b and 7.4c respectively. Below these search distances, and indeed below the average cell side length, problems occur. At a search distance of 3.0E-02, not all of the cells in mesh1 which lie at the centre of the overlap region of mesh2 are found by this search, as is illustrated by the edge plot in Figure 7.4d.



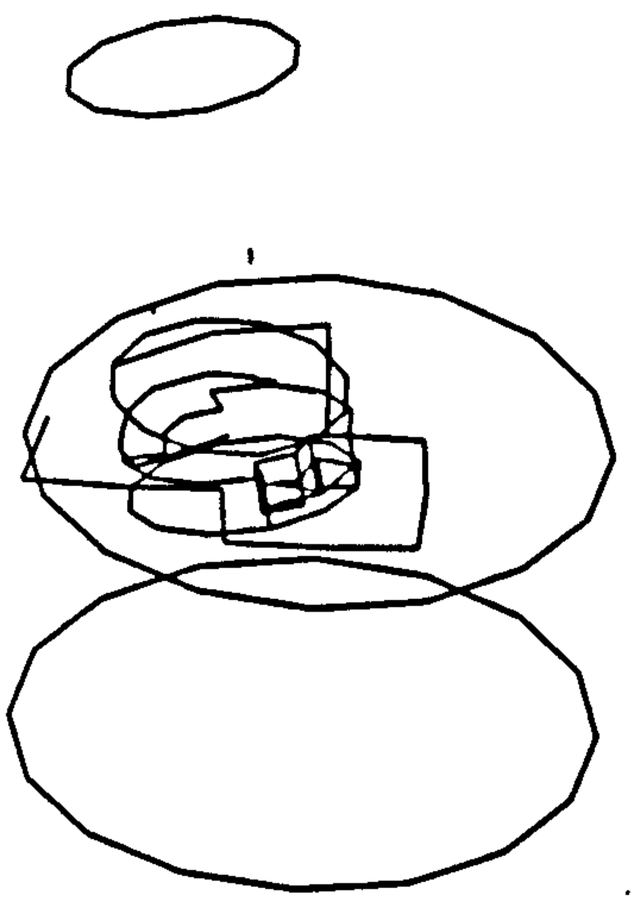
a All cells are tetrahedra



b Smaller search distance



c Smallest acceptable search distance



d Search distance too small

Figure 7.4 Resultant meshes for test case 1

In Figure 7.4d, the outline of some hexahedral cells can be seen in between the two meshes which should have been linked by a coherent mesh of tetrahedral cells. These are cells from the original meshes which lie inside the overlap. However in this case the search distance is too small and so some vertices are not found using this technique, as is illustrated in Figure 7.5.

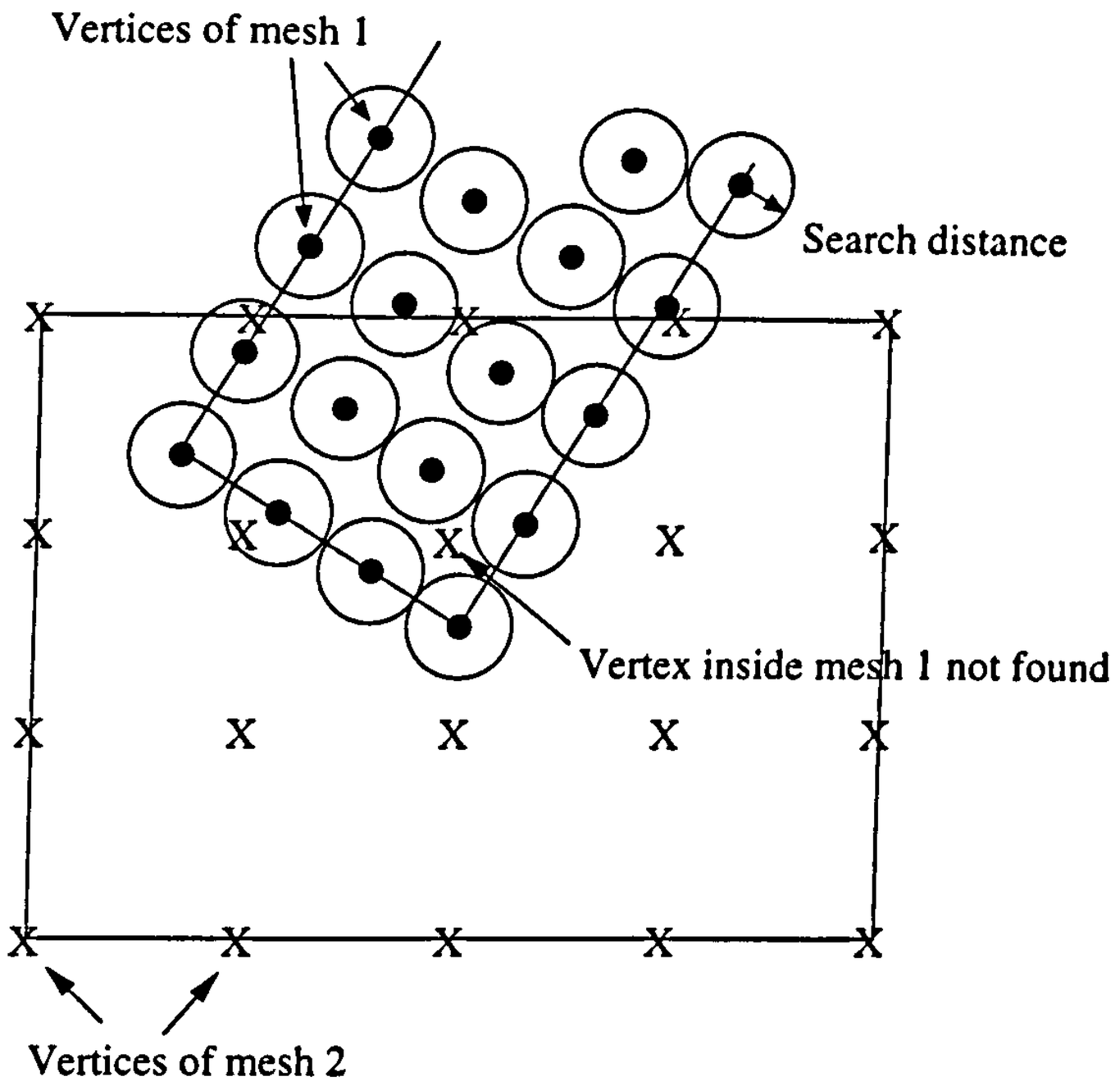


Figure 7.5 Search distance too small for large initial mesh overlap

Thus cells at the very centre of this overlap are not considered to lie between the two meshes which produces this incorrect result. For smaller search distances again, such as 1.0E-02, the mesh construction program simply does not finish. This test case showed that the search distance should be at least half of the maximum distance between the surface vertices.

The run times for the mesh linking of these two meshes with suitable point coincident tolerances and search distances ranges from 2 to 6 minutes actual time, or 0.2 to 0.3 seconds cpu time. Whilst this is a good run time for user interaction in a mesh construction process, the quality of the cells created must also be examined.

The triangular faces of tetrahedral cells which lie on the surface are very important to the final cell quality of the new mesh. This is because no matter how good the internal angles of the tetrahedral cells are made during mesh refinement, their surface faces will not be altered

in remeshing and so the surface face quality will always be the limit of refinement possible. For example, if a surface triangle has a minimum internal angle of 2^o then the tetrahedral cell containing this face cannot be refined so as to make this angle greater than 2^o in remeshing. Table 7.3 shows the maximum and minimum internal angles for the surface faces of the tetrahedra created with the user inputs outlined above. Since the cell quality of the original meshes is entirely dependent upon the user, only the tetrahedra created in the mesh construction process are considered for refinement.

Tolerance	Search distance	Min. angle	Max. angle
1.0E-09	0.20	6.1^o	163^o
1.0E-09	0.10	6.1^o	162^o
1.0E-09	0.07	6.1^o	162^o
1.0E-06	0.07	6.1^o	158^o

Table 7.3 Surface face quality for test case 1

In Table 7.3 it can be seen that varying the search distance whilst keeping the tolerance at 1.0E-09 has no effect upon the minimum internal angle and very little effect upon the maximum internal angle. However changing the point coincidence tolerance to 1.0E-06 and keeping the search distance at 0.07 has a definite effect upon the maximum internal angle. This demonstrates that the point coincidence tolerance affects the surface cell quality, as can be seen in Figure 7.6

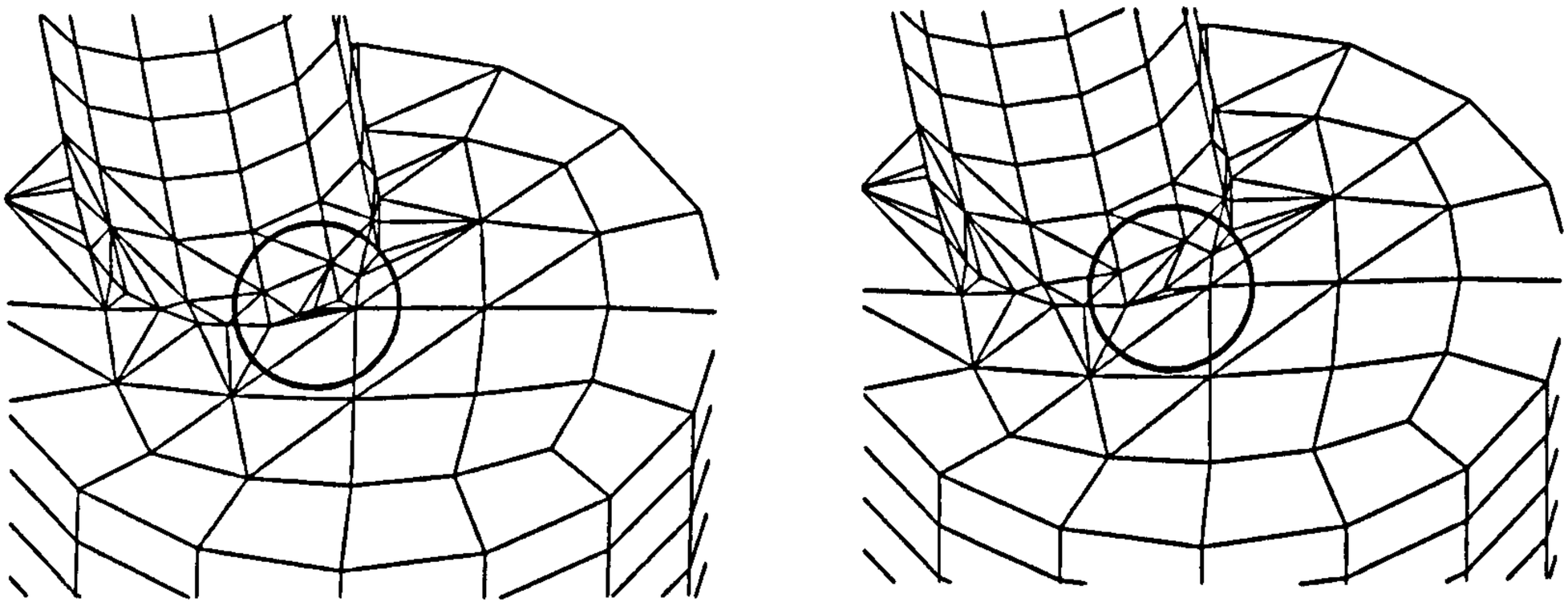


Figure 7.6 Different tolerances produce different surface refinements

The reason for this small effect of tolerance upon surface cell quality is a subroutine which merges adjacent vertices along the intersection once the entire intersection between the two initial meshes has been determined. The subroutine detects vertices lying within a distance of (100 x tolerance) from one another and merges any that are found. The author selected the multiplying factor of 100 based upon the test cases presented in this chapter. A smaller multiplying factor would produce surface angles that are too small whereas a larger factor would produce larger and hence better quality surface angles, but the surface definition along the intersection could be lost. To demonstrate the influence of this subroutine, Figure 6.8 showed the resultant mesh from a run using a point coincidence tolerance of 1.0E-06 and a search distance of 0.07, but where the vertex merging subroutine has been omitted. The large number of poor quality cells along the intersection can easily be seen in this Figure.

The best quality mesh obtained in this test case, as was indicated by Table 7.3, was obtained using a point coincidence tolerance of 1.0E-06 and a search distance of 0.07, and of course including the vertex merging subroutine. The surface of this mesh cane be seen in Figure 7.7.

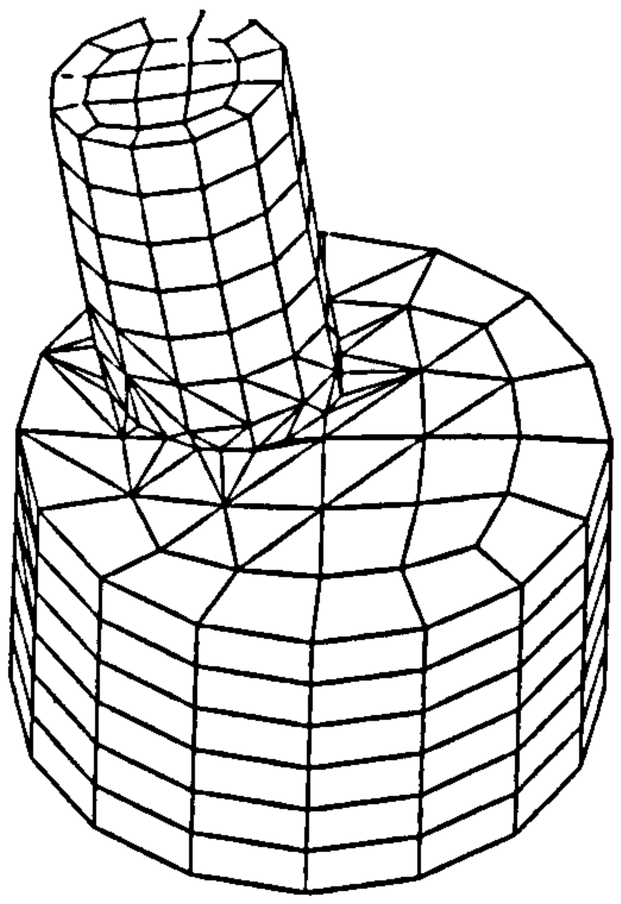


Figure 7.7 Best resultant mesh from test case 1

7.6.2 Joining two brick-shaped hexahedral meshes

Test case two is the joining of two block meshes made up of hexahedral cells which can be seen in Figure 7.8.

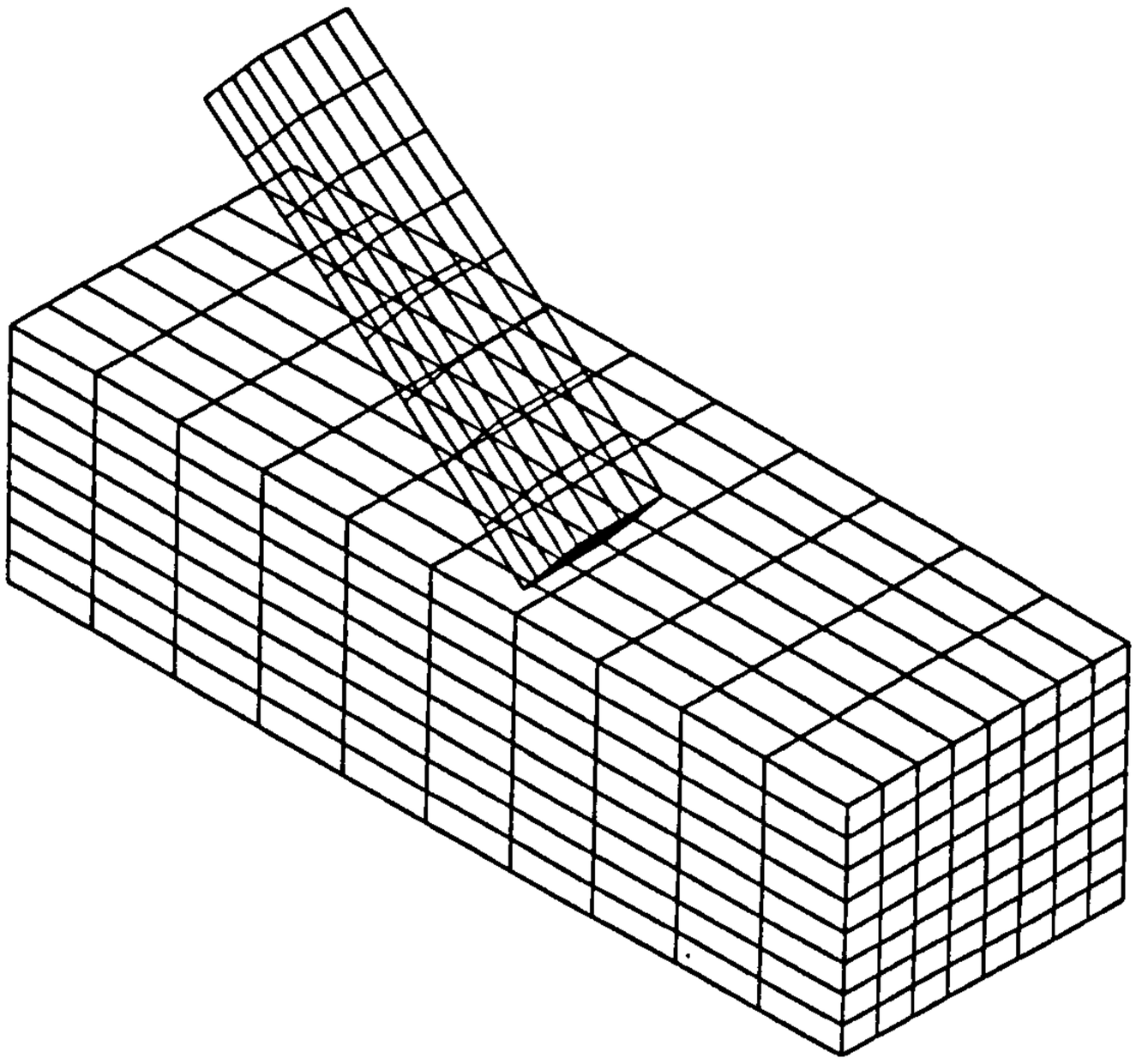


Figure 7.8 Overlapping initial meshes for test case 2

Mesh1 consists of 128 cells with an average cell side length of 0.11, and mesh2 contains 640 cells with an average cell side length of 0.17. As for the previous test case, the effects of the various user inputs are explored. The first to be considered is the point coincidence tolerance. Tolerance values ranged from 1.0E-12 to 1.0E-03 as shown in Table 7.4.

Tolerance	Result
1.0E-12	Error message, program stopped
1.0E-09	Program ran OK
1.0E-06	Program ran OK
1.0E-04	Program ran for 2 days, stopped by user
1.0E-03	Program ran for 2 days, stopped by user

Table 7.4 Point coincidence tolerance for test case 2

This table shows that tolerance values of 1.0E-03 and 1.0E-04 were too large, and once again the program ran for 2 days without finishing. Suitable tolerance values were 1.0E-06 and 1.0E-09, whereas a tolerance of 1.0E-12 was too small and the program reported an error and stopped. A tolerance of 1.0E-06 was therefore used for the subsequent user input tests.

The next input examined is the search distance which ranged from 1.0 to 3.0E-02. A search distance of 1.0 resulted in cells which crossed the surface of the original meshes, as was discussed in section 6.6.2.1. This occurred because mesh1 enters mesh2 at an angle and if too many cells are considered then the central vertex is placed too far from the actual intersection of the meshes. This problem is illustrated for this case in Figure 7.9.

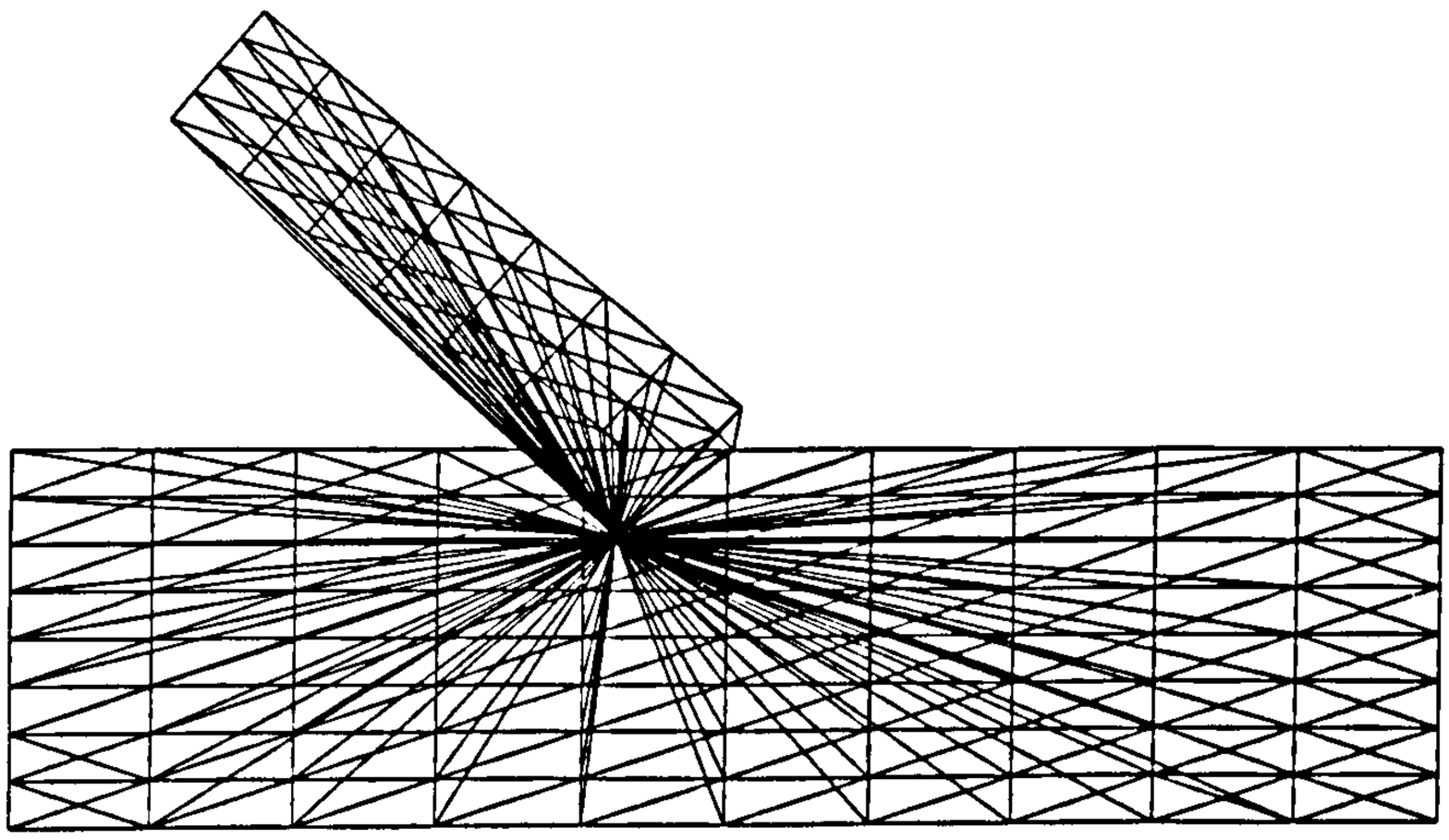


Figure 7.9 Central vertex not at the centre of the overlap region

Search distances of 0.3 and 0.1 were both suitable for this test case and the joined meshes are illustrated in Figures 7.10a and 7.10b respectively. Finally a search distance of 3.0E-02 was attempted, but the program ran for 2 days without finishing.

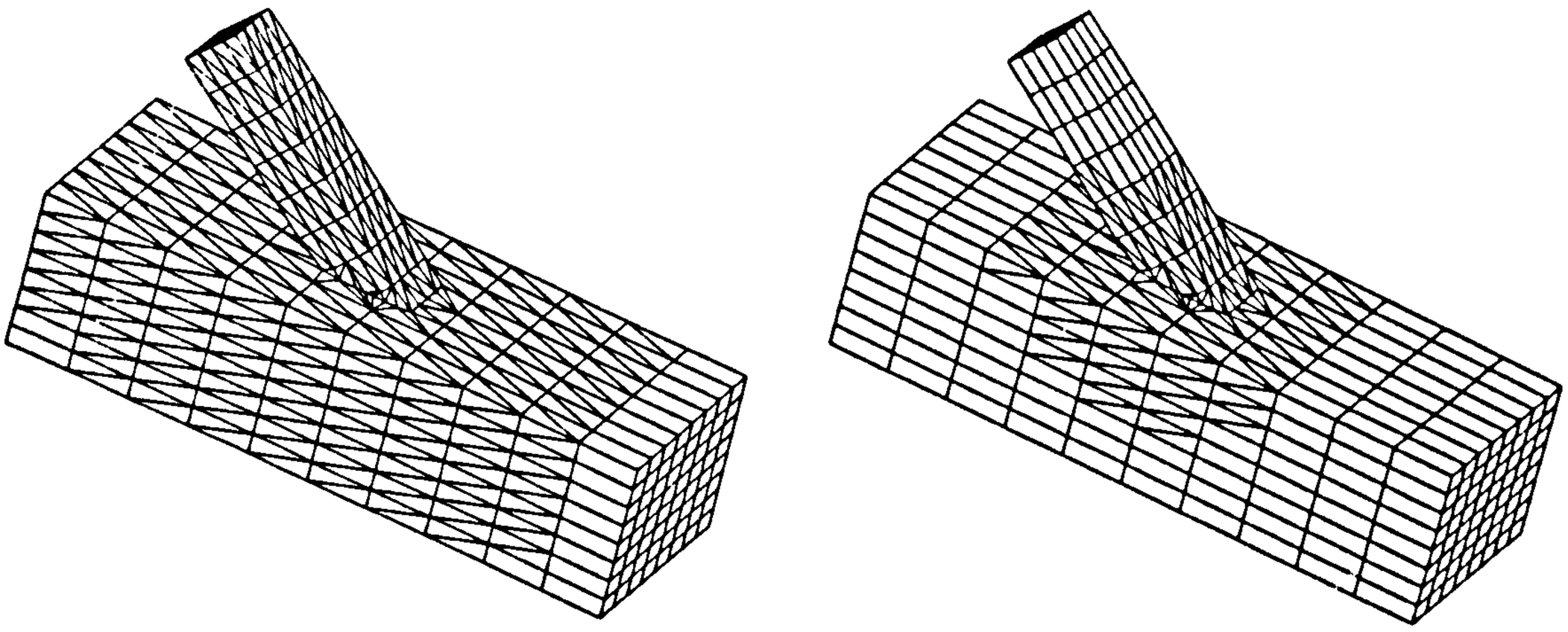


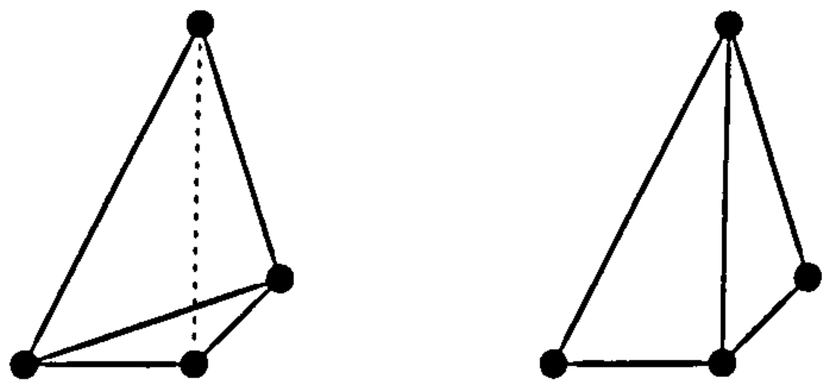
Figure 7.10 Test case 2 for two search distances

Having determined the suitable user inputs for this test case, the quality of the tetrahedral cells created is shown in Table 7.5.

Tolerance	Search distance	Min. angle	Max. angle
1.0E-09	0.01	3.7°	154°
1.0E-06	0.01	7.8°	151°

Table 7.5 Surface cell quality for test case 2

One potential source of problems for this test case test case are the sharp corners at the intersection. Here the corners may be removed when the surface mesh at the intersection is created. Figure 7.11 illustrates how a set of vertices may have two possible surface triangles fitted through them.



Edge not followed Edge followed correctly

Figure 7.11 Two ways of meshing a set of corner points

New triangles are created at the intersection between two meshes. If the only selection criterion for these triangles is the internal angle then the corners will inevitably be cut. In order

to preserve the surface at corners, a second criterion was included such that the new triangles created after an intersection point has been found must lie in the plane of the original triangular face. Since the question of whether a point lies inside or outside of a plane has a 'tolerance' to it, that is, it depends on rounding off numbers, then a planar tolerance of 15^o was chosen. That is, the new triangle is acceptable only if its centre lies a maximum of 15^o out of the plane of the original triangular face. Although this angle is arbitrary, it does ensure that sharp corners are not cut.

The run times for this mesh construction was 0.1 second cpu and under 2 minutes of interactive time. Once again this is an acceptable time for interactive users.

7.6.3 Cutting triangles from hexahedra

For test case 3, a cylinder described by 252 triangles on its surface is cut from a block meshed by 512 hexahedral cells. These initial meshes are shown in Figure 7.12.

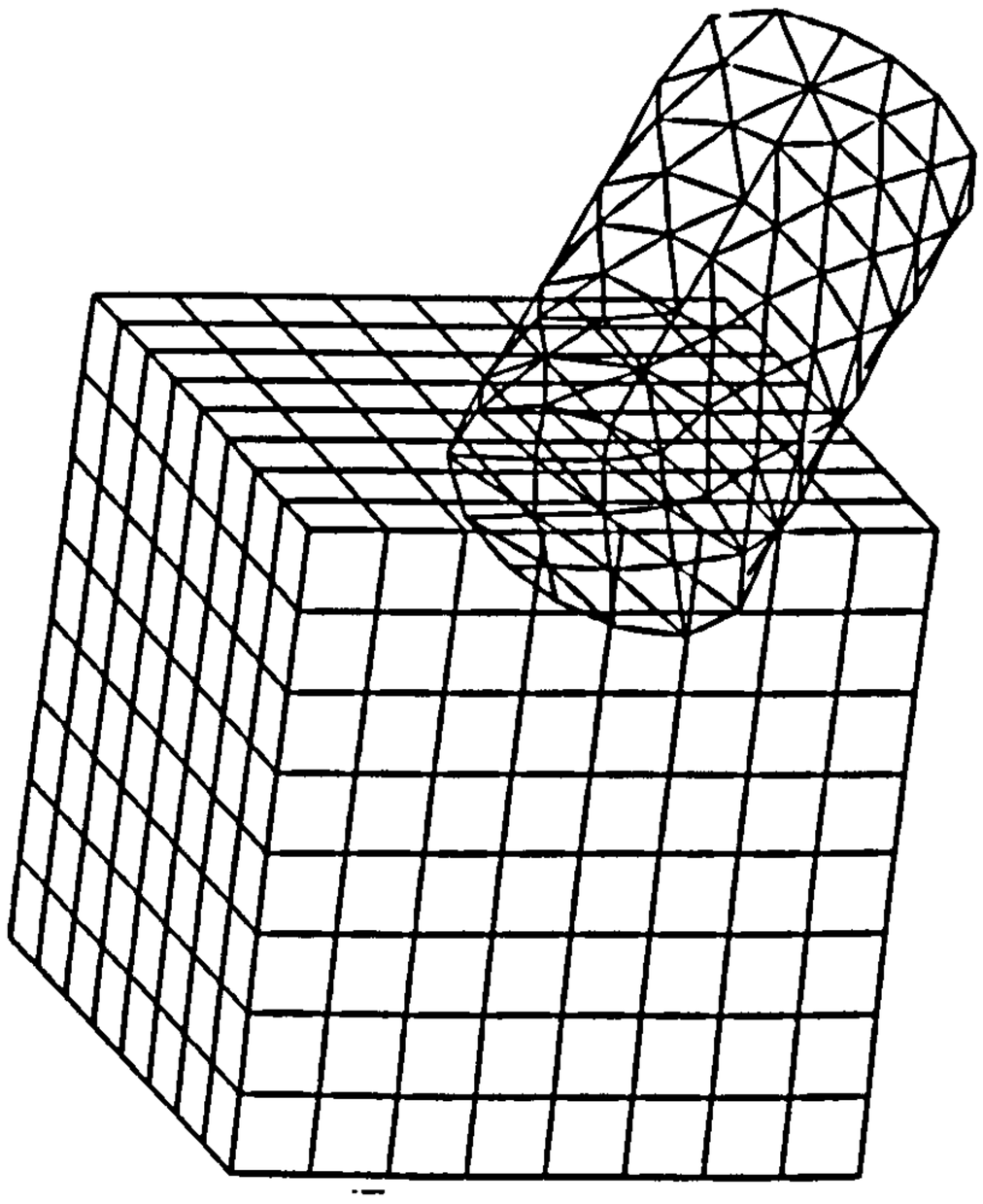


Figure 7.12 Initial meshes for mesh cutting test case

In the first stage of the mesh cutting process, the intersection of the two meshes was found.

Since the cutting mesh will not feature in the final mesh, then it is deleted from the mesh at this stage. The cutting mesh is stored separately, however, so that the surface of the cut mesh may be calculated. The remaining mesh therefore has intersection points linking to the central vertex, effectively creating a new surface. Figure 7.13 shows the surface and the edges of this stage of mesh cutting.

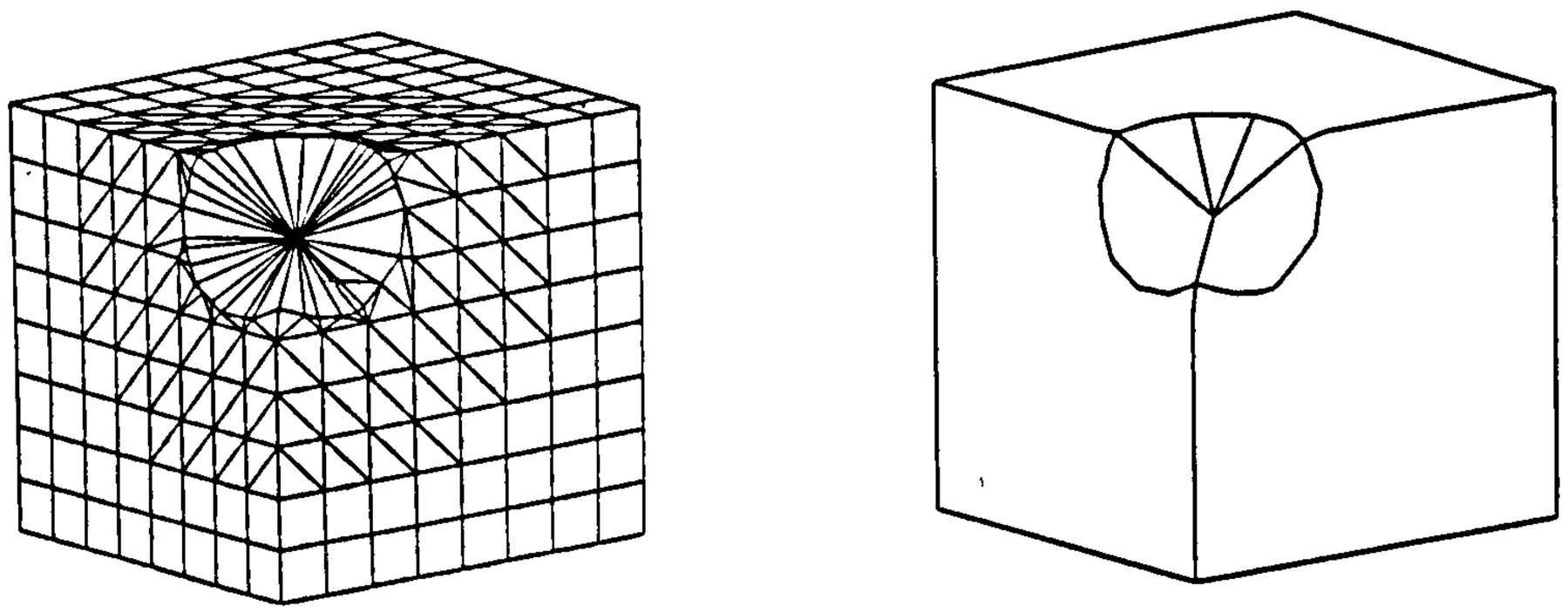


Figure 7.13 Edges of mesh after intersection is found

The next stage is handled in the second program. In this, the actual surface mesh is found and then the surface faces are refined. The surface mesh is found by finding the intersection of all the tetrahedral cells with the faces of the cutting mesh, as was described in section 6.8. Figure 7.14 illustrates the mesh cross-section obtained from this procedure.

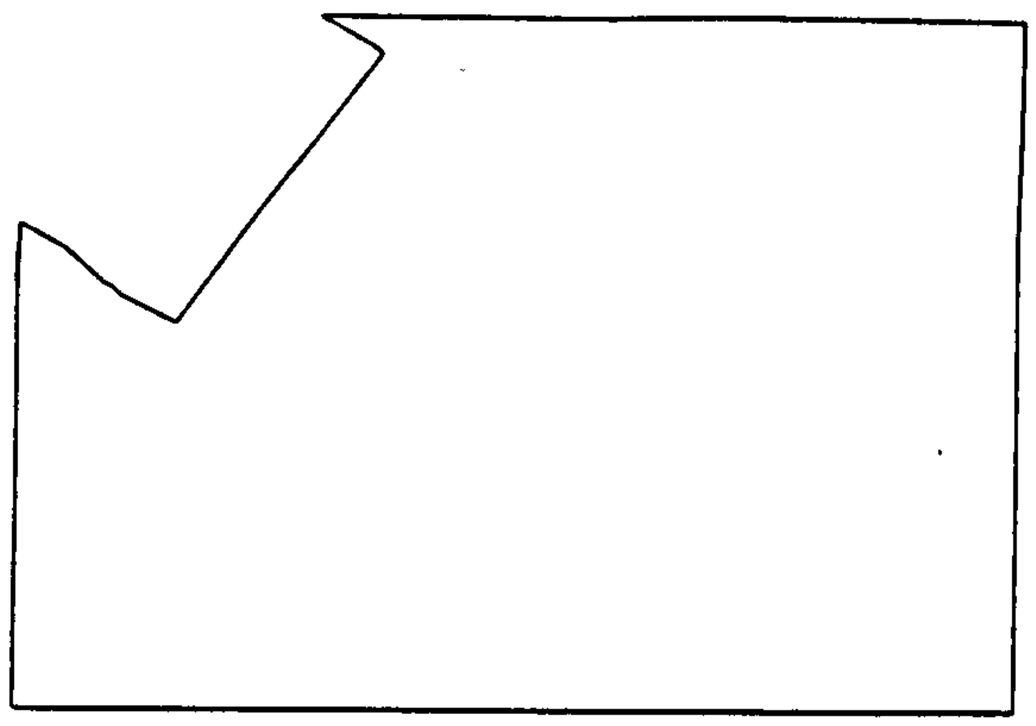


Figure 7.14 Cross-section of cut mesh

The next step is surface refinement which is a two stage process. Firstly all surface vertices that lie within a set distance are merged. In order to preserve the edges of the cutting mesh, merged vertices take on the label and position of the intersection vertices wherever appropriate. The second stage is a search for small internal angles on the surface of the mesh. The angles used are specified by the user, whereas the merge distance is set within the program to be 1% of the range of surface edge lengths found.

Variations in both the merge distance and the user-input minimum angle were explored for this test case, and the results are presented in Table 7.6 below.

Merge angle	Merge distance	Min. angle	Max. angle
5.0°	2.0%	1.3°	170°
10.0°	2.0%	1.9°	170°
15.0°	2.0%	2.0°	170°
30.0°	2.0%	1.3°	170°
10.0°	5.0%	1.0°	171°
10.0°	10.0%	1.0°	171°

Table 7.6 Variations for cut surface refinement

It can be seen from Table 7.6 that the variables for surface refinement have little effect upon the extreme angles of the surface faces. However, plots of the cells at the surface shown in Figure 7.15 shows that the surface cell quality is significantly influenced by these variables.

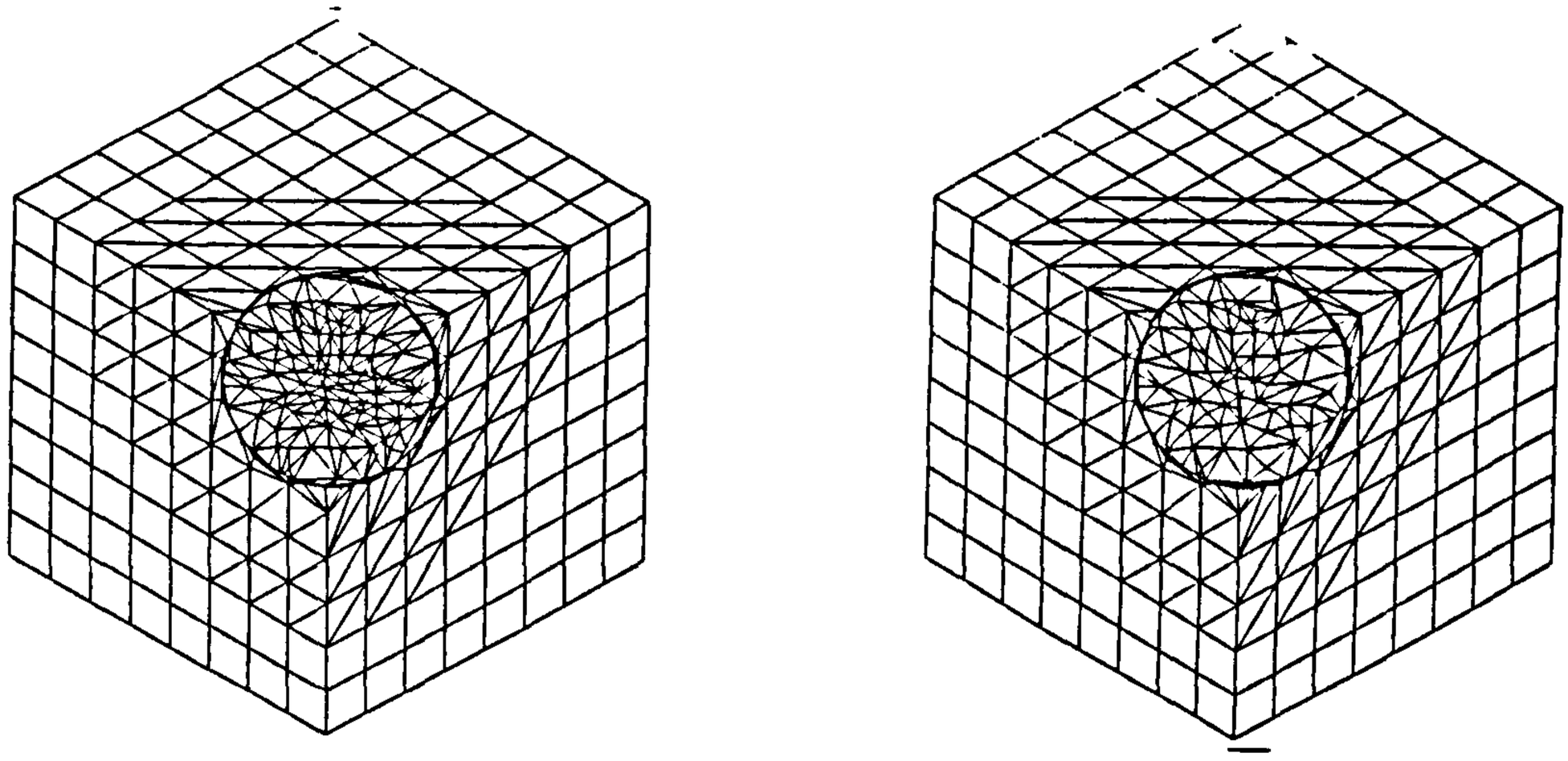


Figure 7.15 Altering surface cell quality

This suggests that whilst most of the surface faces are refined according to the variables input by the user, some highly distorted cells are not refined.

There are two possible explanations for these results. Firstly, when the cell refinement occurs, the quality of the newly created cells are not considered. In this case, the refinement would have to be continued iteratively until no further cell refinement was required. This iterative refinement was attempted, but the same highly distorted cells existed after the extra refinement. The other explanation for these results is constraints upon the refinement subroutines. For example, cells with vertices lying on the original intersection line are not refined in order to preserve the mesh-mesh intersection. If these cells were refined, then the good quality intersection may be lost. A further examination of the refinement of these cells is recommended for further study.

7.6.4 Internal cell refinement

The fourth test case features two meshes which are more representative of industrial applications. Mesh 1 represents an engine intake port and contains 2640 cells. Mesh 2 represents a flat roof combustion chamber and contains 16800 cells. The two initial meshes are shown in Figure 7.16.

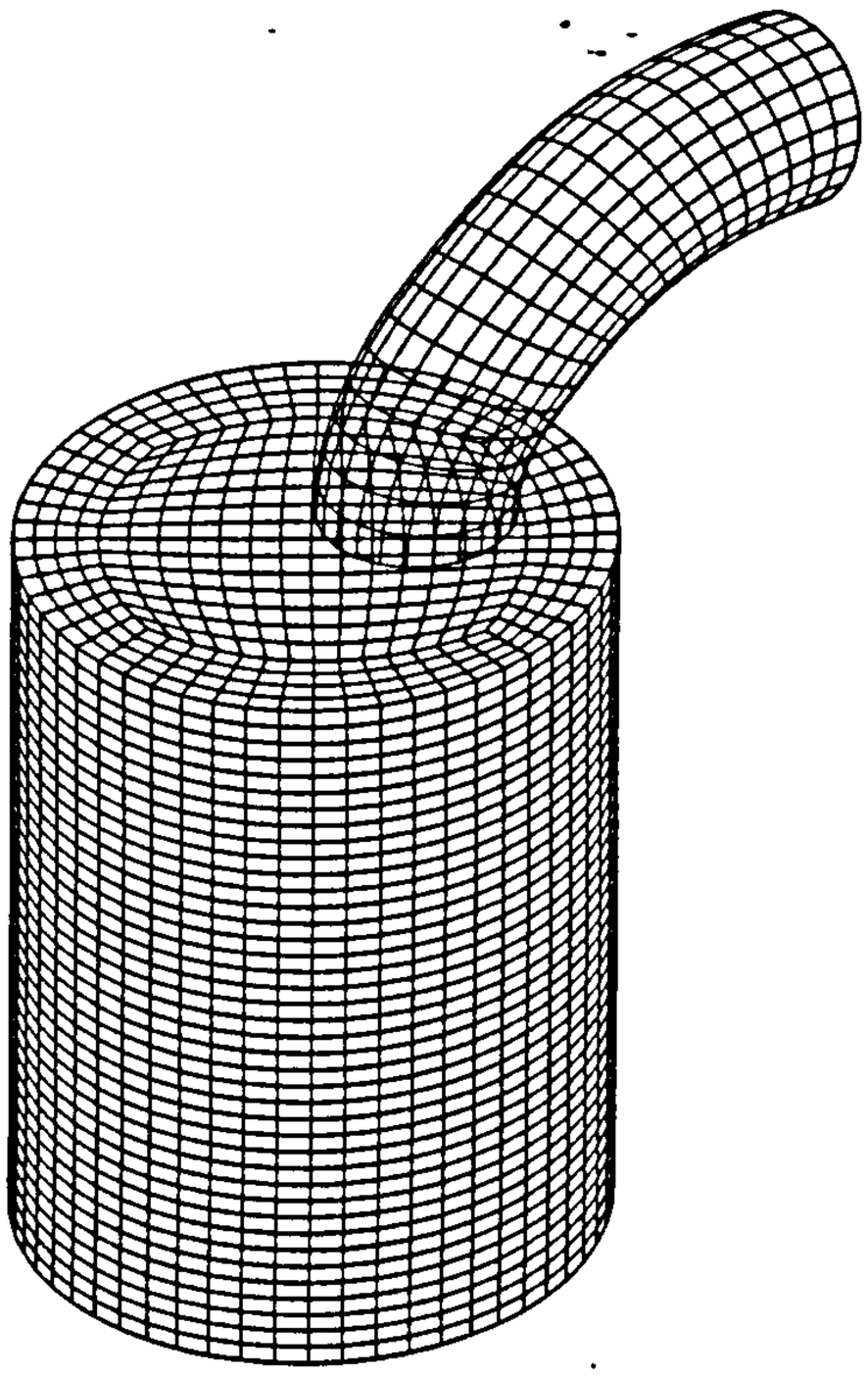


Figure 7.16 Initial overlapping meshes for test case 4

The average cell side length for these meshes was 1.0 and a point coincidence tolerance of 1.0E-04 was found to be suitable. Test case 2 has already demonstrated that a search distance of the same order of magnitude as the cell side length may be suitable. In addition, the curvature of mesh 1 had to be accounted for, as section 6.6.2.1 explains, too large a search distance could create crossed faces. Thus a search distance of 1.1 was used which is a little larger than the cell side length. With the above variables, the mesh linking in the first mesh construction program took 855 seconds cpu, approximately 20 minutes run time. The surface of the resultant mesh is shown in Figure 7.17.

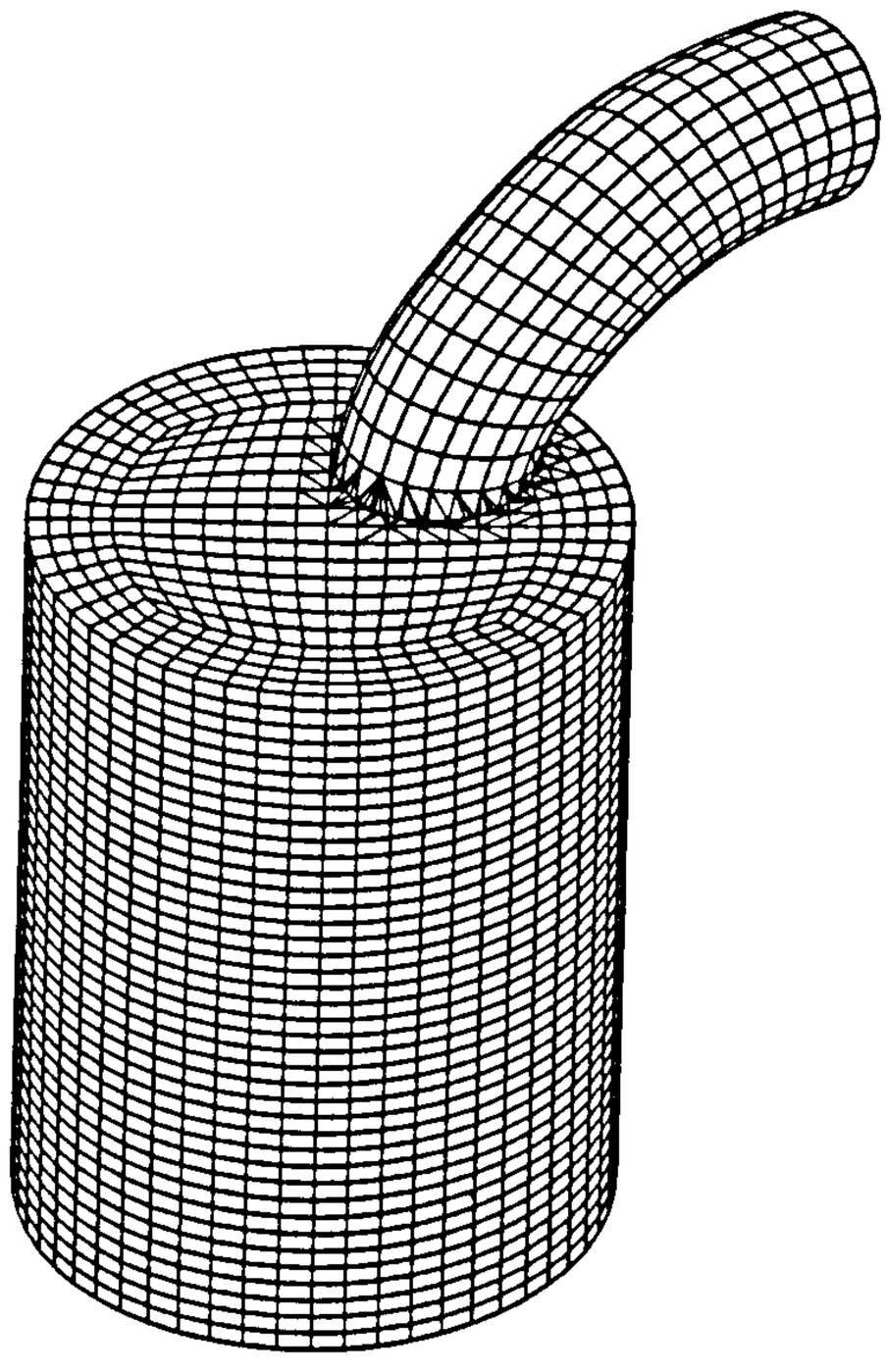


Figure 7.17 Surface of joined mesh

The second mesh construction program remeshed the internal tetrahedra of the join region. However additional tetrahedra were also included in the mesh to blend the all tetrahedral mesh to the rest of the joined meshes. Therefore these blending tetrahedra were ignored and only those linked to the central vertex were considered. This program took 880 seconds cpu, which was approximately 20 minutes run time.

As described in Sections 6.6.2.2 and 6.6.2.3, the remeshing consisted of edge swapping and vertex insertion. To illustrate the effects of each of these processes, Figure 7.18 shows the internal angle of each face in the mesh graphically. Since the surface faces are not changed in remeshing, they are not included in these graphs. This means that Figure 7.18 shows the internal angles of the triangular faces inside the mesh only.

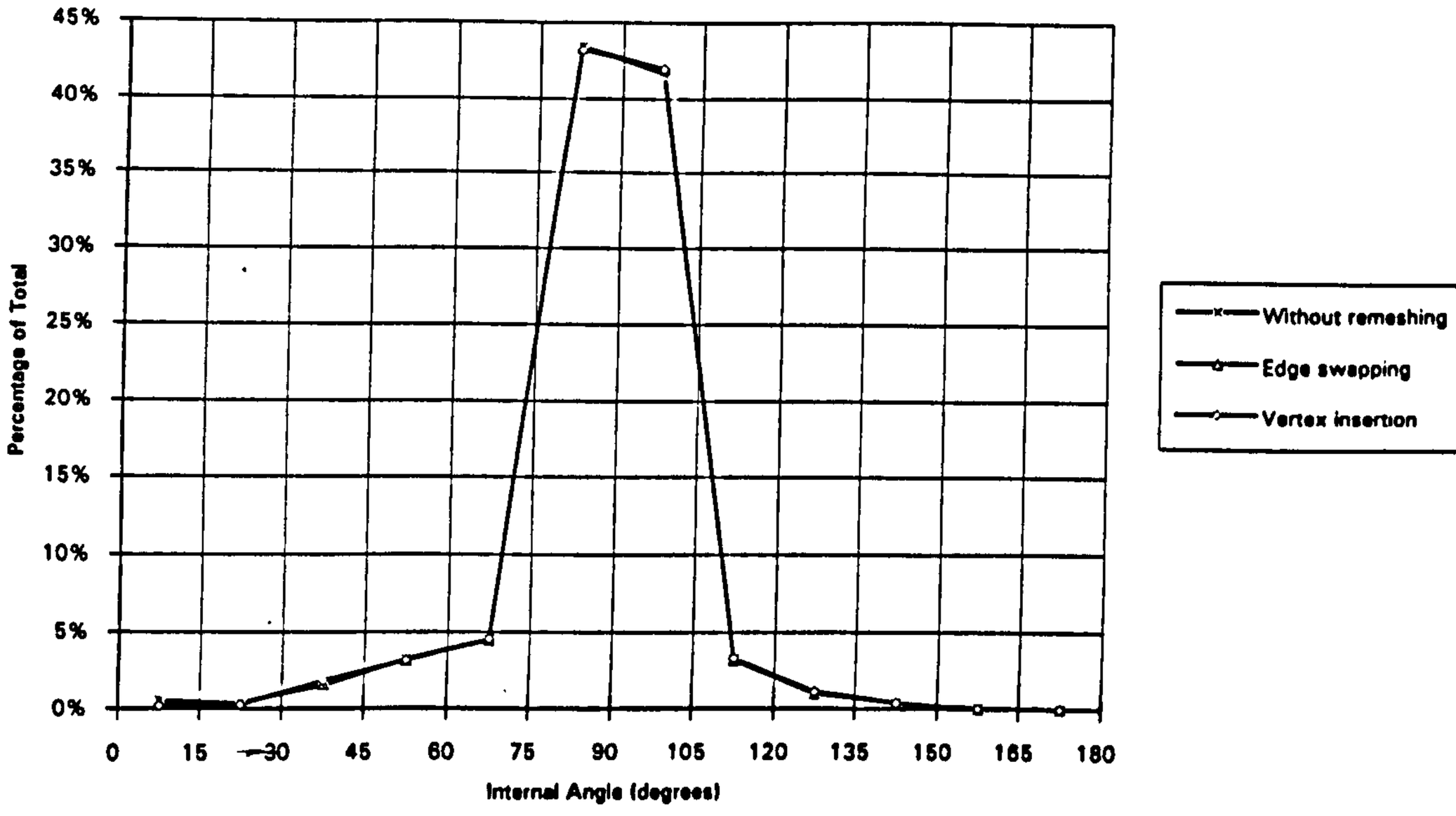


Figure 7.18 Internal angles of triangular faces in remeshing

Figure 7.18 shows that some mesh refinement is achieved by vertex insertion and edge swapping. However some of the internal angles remain very small. This is probably because the edge swapping routine will omit a group of cells of their newly created faces between the equatorial nodes lie in the same plane as existing faces between these nodes. This situation is illustrated in Figure 7.19.

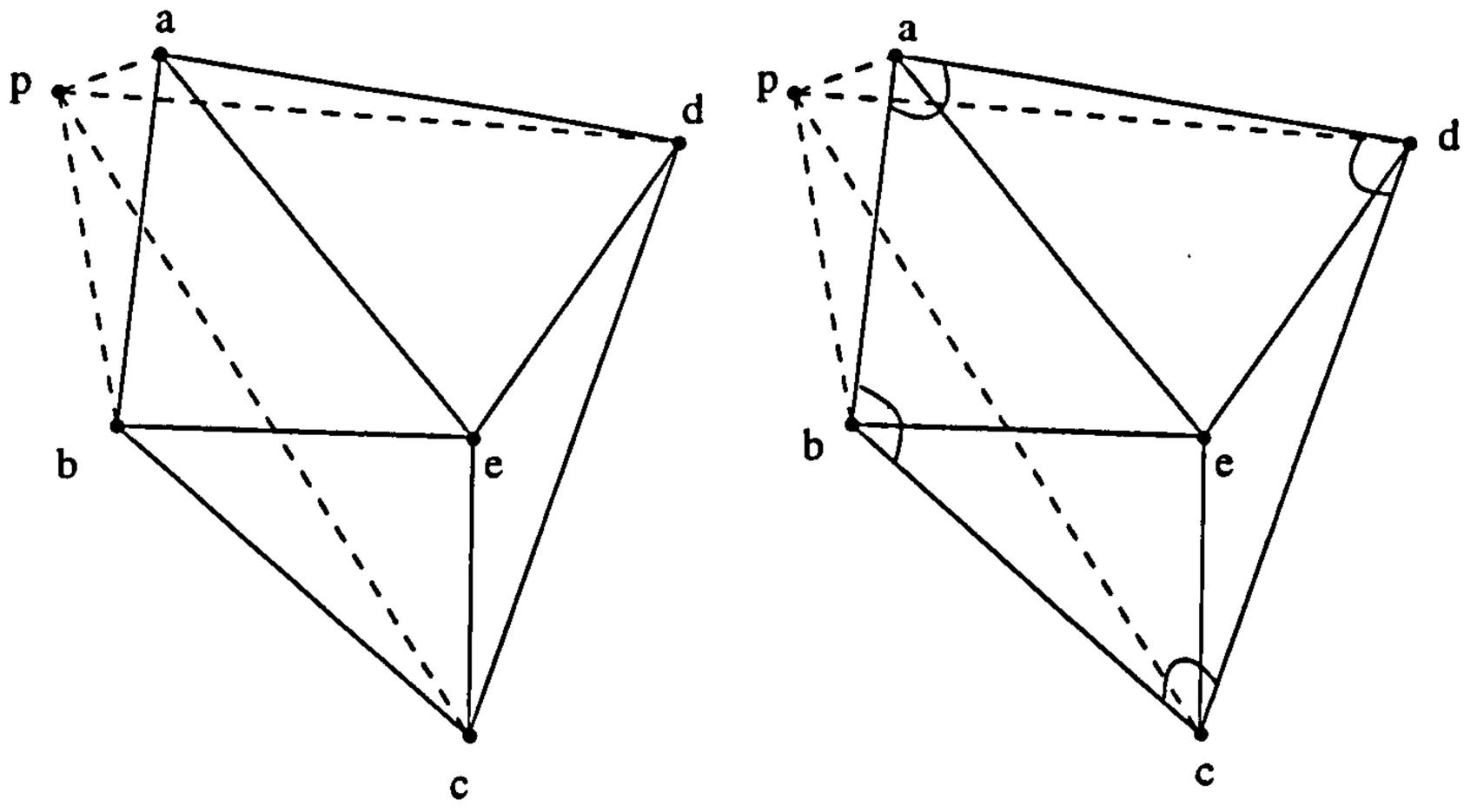


Figure 7.19 Cell groups omitted by edge swapping process

In Figure 7.19, four faces containing the vertices a,b,c,d,e are linked to the central vertex p, and vertices a,b,d,e lie in the same plane. The check for smallest equatorial angle finds angle subtended at vertex a to be the smallest. Thus a new triangular face should be created using the equatorial vertices a,b,d. However this internal face would in fact lie in the plane of vertices a,b,d,e and so it would directly overlap the external faces a,b,e and a,d,e. The edge swapping subroutine recognises this situation and so does not perform the edge swapping for these faces.

If the edge swapping process can be improved then remeshing will certainly be very successful. However alternative mesh generation techniques may also be used. For example, the advancing front method will also create a tetrahedral mesh inside a shell of triangular faces. Both techniques would have to be compared to determine which method is faster, uses the least computing resources and produces the best quality cells.

7.7 Additional case

In addition to the four test cases run, one further examples was attempted. These examples

explored mesh construction, in particular, mesh cutting, for more industrially relevant situations, which is a simple aerodynamics research model, the Ahmed model, in a wind tunnel.

A simple mesh of the Ahmed model, shown in Figure 7.20, is cut from a block mesh with the dimensions of a model scale wind tunnel.

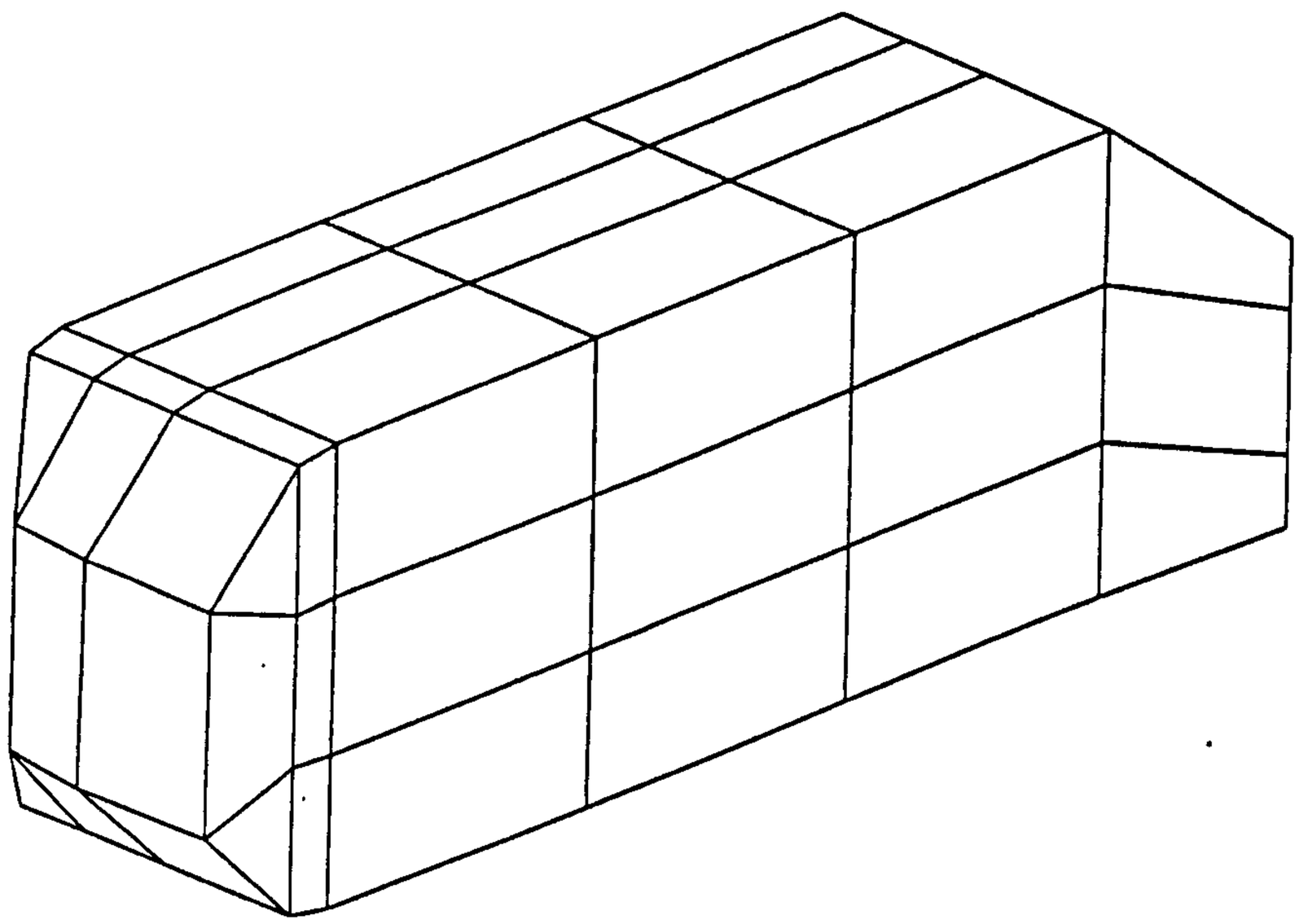


Figure 7.20 Simple surface mesh of Ahmed model

The surface intersections took 510 seconds cpu, about 12 minutes run time. The surface mesh quality was good with internal angles ranging from 11^{o} to 171^{0}. The results of this surface intersection are shown in Figure 7.21.

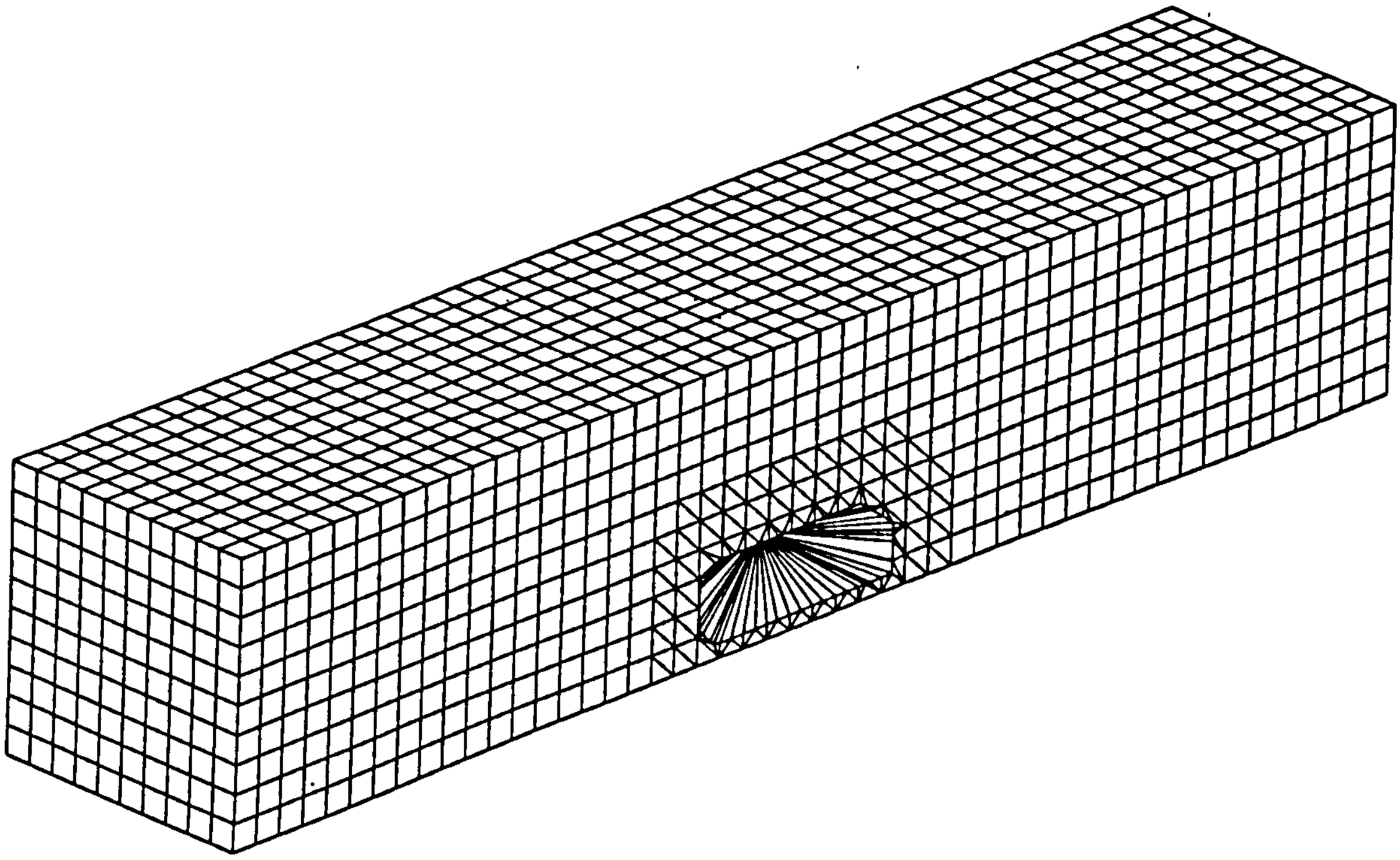


Figure 7.1 Surface intersection for Ahmed mesh in wind tunnel mesh

This Figure shows that the surface intersection works very well however problems arose at the inner edges of the mesh near the surface of the Ahmed model, which were not defined clearly. This suggests that the mesh cutting procedure needs further development before it is robust enough for a range of mesh cutting situations

7.8 Future program developments

The scope of the mesh construction program and its application have been discussed in this chapter, and possible future developments have been mentioned. In order to make the program robust for industrial use and even for commercial exploitation, the following developments are recommended by the author;

1. Rewrite the program with computational efficiency and speed in mind. This would include

both faster search algorithms and variable array dimensions which can now be used with the FORTRAN 90 programming language.

2. Use the automatic inside/outside tests already installed in the program to perform the search for overlapping cells. This means that the user will not have to specify a search-distance, which would make the program more fully automatic.

3. Systematically study the influences of the point coincidence tolerance for every subroutine in which it is used. This will produce more concrete guidelines for the user, and may even lead to an automatic selection of the tolerance.

4. Expand the file translator to read and write a range of data formats from the most popular propriety mesh generation codes. Also, remove the requirement to have the vertices and cells labelled from 1.

5. Examine possibilities for further refining the surface of cut cells along the intersection.

6. Explore other methods of internal cell refinement for the tetrahedral cells in order to obtain an optimum mesh quality quickly. This may include using another tetrahedral mesh generation technique, such as the advancing front method.

7. Improve the flexibility of the mesh cutting part of the program, particularly to account for sharp corners on the surface of the cutting mesh.

7.9 Summary of chapter 7

The mesh construction program is both flexible and easy to use. It is a novel technique which can significantly increase the speed and ease of mesh generation for complex geometries when used in conjunction with other mesh generation tools. CFD solvers which can use

meshes of mixed cell types have recently become available and so the mesh construction has practical applications to many aspects of CFD analysis.

A number of test cases were performed examining the influence of input variables on the quality of mesh produced. The surface intersection part of the program was shown to be robust for a number of test cases, however the internal mesh refinement and cutting operations work only for limited cases. Therefore substantial development is still be required before the program may be used with confidence in an industrial setting.

8.0 CONCLUSIONS

8.1 Introduction

The work presented in this thesis is discussed and summarised in this chapter. First a summary of the model creation and mesh generation processes is given in Section 8.2. A discussion of the integration of these methods into the engine design process at Jaguar then follows in Section 8.3. Here recommendations are about this integration into current working practices. Section 8.4 summarises the mesh generation problems for using CFD in engine design, along with a discussion of the author's own mesh construction technique. Finally, the scope for future developments is discussed in Section 8.5.

8.2 Times to build and solve

One objective for this work was to determine a quick and easy method for creating CFD models using I-DEAS and STAR-CD software. The initial prospects looked promising since I-DEAS supported both solid modelling capability for rapid model creation as well as an automatic hexahedral mesh generator. Problem arose, however, when STAR-CD v1.4 could not accept the I-DEAS generated mesh and then SDRC withdrew the automatic hexahedral mesh generator in I-DEAS V due to quality concerns. Thus alternative methods for creating the CFD model had to be used.

Using CAD data supplied by Jaguar, three computer models were created in I-DEAS using solid models either for part or for all of the model. These models were then meshed in a number of ways, using either the I-DEAS automatic tetrahedral mesh generator or I-DEAS mapped mesh volumes. The best quality and therefore most suitable mesh was found to be that created from mapped mesh volumes. The computer model for this mesh was created using CADDS curves for the port, but the rest of the model was built from the wireframe geometry derived from solid models of the roof, valves and cylinder defined and created in the FEM module.

Thus the best mesh for the CFD analysis using STAR-CD was the one which took the longest to build, and indeed the most user skill and experience. In total, the mesh was ready for CFD analysis after three weeks, although a more experienced I-DEAS user could have generated the mesh in about two weeks. Once generated, the CFD solution for one valve lift took 120 cpu hours, which was effectively one week on the CONVEX at Jaguar. Two more valve lifts were analysed for the engine geometry, and so the total time for the CFD model creation and analysis was about 6 weeks.

This falls far short of the desired overnight runs which would be most useful for engine design. Consequently CFD could not be used in the initial design stages when a number of ideas could be tested quickly. This does not rule out CFD's usefulness to engine designers, however, and CFD could still be instrumental in fault diagnosis in the early stages of engine development.

However the future use of CFD in the initial stages of engine design still holds some promise. Computer power is changing constantly, with faster processors, which could significantly shorten run times, along with a reduction in the price of such computers. In addition, research into mesh generation techniques continues to look into faster and more automatic methods. The problems of using CAD data in different software packages is leading to improved CAD descriptions and to CAD 'translators' which can automatically correct typical CAD problems of surface mismatching.

8.3 Results comparisons

The CFD results presented in chapter 4 showed a fair correlation to experimental measurements taken using LDV techniques on a specially built engine rig with optical access. This correlation was obtained despite some drawbacks to the model.

Firstly, a problem was encountered when the CFD model for mesh-3 did not match the experimental rig. The cylinder was not extended and the valve lift had been wrongly measured in

the CFD model. For any validation work it is essential that the computer model and experimental rig are as close to the same geometry as possible. In this work, mesh-4 was subsequently created and a better correlation was obtained.

A second problem was that a symmetric flow field was assumed in the CFD analysis, based upon the fact that the engine geometry was symmetric. This is, of course, not necessarily true and indeed the experimental results showed an asymmetric in-cylinder flow field. This issue is discussed further in Section 4.5, from which it is clear that the CFD results for a steady flow case might not predict an asymmetric flow field even if the entire geometry was analysed. Thus this is simply a potential source of error which should be considered when interpreting and validating CFD results.

The mesh used in this work was very coarse, about half the number of cells than ought to have been used for a detailed analysis. Whilst this was about the maximum mesh size that could be run using Jaguar's computer resources, ideally a finer mesh should have been used. The mesh density was then another source of error in the CFD calculations.

Despite errors and approximations in CFD results, the essential question is whether the CFD results give the engine designers the information that is required. As discussed in Section 8.2, the analysis was too slow to be useful for testing a wide range of engine designs at the concept stage. The information provided in this case, however, proved useful in the evaluation stage of the engine's development. Here a flow related fault in the engine geometry was identified through the flow visualisation, which clearly showed a strong flow down the far walls of the cylinder. The fuel therefore condensed onto the walls which explained why fuel was seen to leak from the bottom of the cylinder in tests of the early prototype.

The next question for the engine designers is how reliable are the CFD results. Evidently validation of the results can be obtained using a specially built optical access engine, but building the rig, taking the measurements and correlating them to CFD would prove very costly for every design modification and take much longer than obtaining the CFD results

themselves. Thus a working solution is to validate the CFD results for one geometry and then to use that 'estimate of error' when predicting the flows through very similar geometries. This approach carries a risk that the geometry has deviated too far from the validated case for the error estimates to be appropriate. In these cases, the skill and experience of the CFD analyst is very important, but it also emphasises the way in which CFD can be used for such cases, that is as a general guide to the flow trends only and not an absolute measure of the flow variables.

In summary, the CFD results presented in this thesis were good enough to be used as a guide to the trends in an engine's design but great caution should always be used regarding the accuracy of unvalidated CFD predictions.

8.4 Integrating CFD into Engine Design at Jaguar

Recommendations to Jaguar for using I-DEAS and STAR-CD were presented in Sections 2.10 and 4.8. The integration of CFD into the engine design process involves more than following these recommendations, however. In summary, the draughtsmen should create the CAD models with CFD and FE analysis in mind. This will minimise the amount of repetition when creating the computer model and will also mean that the CFD users will not need to be so highly skilled in I-DEAS modelling as would otherwise be the case. Secondly, I-DEAS solid modelling should be used to create a complete model of the geometry and the mesh should then be generated using mapped mesh volumes in I-DEAS.

Both I-DEAS and STAR-CD have proved to be suitable for the CFD analysis, although less useful than originally anticipated. Since computer software is continually being developed, it is strongly recommended that regular software assessments are made. This will ensure that Jaguar uses the tools most suitable to the CFD analysis required. For example, chapter 4 presented the first steps of transient CFD analysis, for which STAR-CD was entirely appropriate. If a further extension includes, for example, combustion modelling, then another CFD code survey should be done to find the most suitable code for this work.

The management of any change in software would also have to be weighed up, however. This would include the retraining of personnel to use the code, the time required for them to be fully experienced with the code, and the possibility of hiring new analysts for specialist applications who are already trained in using this software. Furthermore, the costs of changing software would have to be considered since many code vendors offer discounts for long term users. Different hardware requirements may also arise, both out of changing codes or indeed when a new version of the software is released as was seen in this work for I-DEAS.

Jaguar's existing hardware available for CFD is a VAX for pre- and post-processing and a CONVEX for the analysis runs. This configuration is not suitable for regular CFD analysis for engines, which involves long processing times and large amounts of computer memory. Indeed only three CFD models could be kept on Jaguar's system at any one time during the course of this work due to memory restrictions. Since the CONVEX is also a multi-user machine, the analysis runs proved to be very disruptive for the other users in Jaguar, which meant that other areas of Jaguar's business was hindered. It is therefore strongly recommended that dedicated high-powered workstations are used for the CFD analysis work. These machines can be linked via the network so that information is easy to transfer, but they can also be left to perform lengthy analyses independently from the other system users. The use of workstations also makes it easy for Jaguar to expand its CFD capability since the addition of an extra workstation does not affect the loading on the multi-user machines. The development of faster and more powerful computers continues unabated. Whilst the mesh generation time remains at about two weeks for such a model, a fast workstation such as an HP735 can produce a solution in about 5 hours in 1994. This gives a total turnaround time of two working weeks for the CFD analysis, with up to 11 valve lifts being analysed over the weekend. Indeed, such faster calculation times also make transient analysis a more realistic option.

Whilst the author recommends that CFD is used in Jaguar's engine development programmes, the establishment of an in-house capability must be carefully considered. This work has shown that CFD cannot be used at the concept stage yet, due to both hardware and software considerations. However CFD can be used in the evaluation stage as a fault diagnosis tool.

As a result, it is anticipated that CFD will not be used intensively for a large number of different geometries and so the cost of establishing an in-house capability may not be justifiable for such low usage. The author would therefore recommend that external CFD consultancy is used in the short-term or that Jaguar's existing CFD personnel are used for the infrequent engine analyses required. For the latter case, dedicated workstations would still be required so as not to disrupt the multi-user systems. This approach will allow CFD to become a regular part of the engine design process. Regular code surveys and hardware assessments would show a point at which the associated costs for an in-house capability dedicated to engine analysis is viable. This does, of course, assume that CFD codes and computer technology will continue to improve with a corresponding reduction in prices as has been the trend for the last twenty years or so.

8.5 Mesh construction

The second focus of this thesis has been the development of the author's mesh construction program. The principal aim of this part of the work was to make mesh generation faster and easier for complex geometries, of which engines are an important example. The test cases outlined in chapter 7 have shown the program to be successful on a limited basis. The program is not at a stage at which it could be released commercially, but it does provide a very promising technique which can be used now. The main problems in its current, prototype form are the levels of users interaction required and its actual use for CFD. The correlation between the point-coincidence tolerance and the results obtained are not clearly visible to the user and so the user must make a number of guesses before a suitable figure is chosen. The author can only provide a guide for the correct order of magnitude at this stage. Tolerance levels notwithstanding, the test cases have shown the technique is versatile for both joining and cutting meshes. This means that its objective of making mesh generation faster by reducing the complexity of the meshes generated has been met.

When work began on this program, Computational Dynamics stated that STAR-CD could be

used with a range of cell types. It later came to light that non-hexahedral cells could only be reliably used in areas where the gradient of the flow variables was small and should not be used at all at the mesh walls. Since the mesh construction program generates tetrahedral elements where the mesh overlap, and where high flow gradients typically occur it means that mesh construction cannot be used in conjunction with STAR-CD. Other CFD codes may well be capable of using the meshes generated by mesh construction, however, one such code being FLUENT-Unstructured [80]. This code uses a mixture of cell types and allows tetrahedra in regions of high flow gradients. Indeed the vendors believe that the main advantage of this code is the flexibility it gives for mesh generation [81].

Further development of this program would enhance it significantly. First of all the data structures could be optimised to make the program efficient in terms of computer memory and also to make it run faster. One possibility is writing it in FORTRAN90 programming language which allows array sizes to be allocated as the program is running and not beforehand as is currently the case. This will mean that the program only uses as much memory as the mesh requires. Furthermore, other methods of filling the shell with tetrahedral elements may also be explored in future versions of this program. In particular, there are a number of ways to refine the initial mesh using vertex-insertion, cell splitting and combining and smoothing. The author believes that improvements in this area will have a significant effect upon the cell quality of the mesh.

The strongest aspect of the mesh construction program was shown to be the calculation of the surface intersection which is now robust for a number of mesh geometries. The internal cell refinement has to be improved and could well be replaced by another more successful method for meshing concave regions bounded by triangles, such as the advancing front method. Similarly the mesh cutting process has not been effective for all cases attempted and could also be improved.

Appendix A

A.1 Intersection of a line and a plane

The intersection of a line and a plane is outlined below, taken from [82]. A plane can be denoted by the equation

$$Ax+By+Cz+D=0 \tag{A.1.1}$$

where A,B,C,D are coefficients and x,y,z are variables in the Cartesian coordinate system.

A line passing through the points $P_1=(x_1,y_1,z_1)$ and $P_2=(x_2,y_2,z_2)$ can be represented by the equation

$$x-x_1=y-y_1=z-z_1 \tag{A.1.2}$$

with

$$\cos\alpha=\frac{(x_2-x_1)}{d} \tag{A.1.3}$$

$$\cos\beta=\frac{(y_2-y_1)}{d} \tag{A.1.4}$$

$$\cos\gamma=\frac{(z_2-z_1)}{d} \tag{A.1.5}$$

where

$$d=(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2 \tag{A.1.6}$$

Thus the point of intersection of the plane and line is;

$$x_i=x_1-t\cos\alpha \tag{A.1.7}$$

$$y_i=y_1-t\cos\beta \tag{A.1.8}$$

$$z_i=z_1-t\cos\gamma \tag{A.1.9}$$

where

$$t=\frac{Ax_1+By_1+Cz_1+D}{A\cos\alpha+B\cos\beta+C\cos\gamma} \tag{A.1.10}$$

This formula for an intersection point has the limitations that; If

$$A\cos\alpha+B\cos\beta+C\cos\gamma=0 \tag{A.1.11}$$

then the line is parallel to the plane. But if

$$A\cos\alpha+B\cos\beta+C\cos\gamma=0 \qquad \text{(A.1.12)}$$

and if

$$Ax_1+By_1+Cz_1=0 \qquad \text{(A.1.13)}$$

then the line lies in the plane. In terms of the mesh building program, if the line is parallel to the plane, but does not lie in it, there is simply no intersection of the line with that face. If, however, the line lies in the plane, then there are an infinite number of intersection points, and this case is treated as if no intersection point has been found. Indeed, such a case will be found only where two meshes touch one another rather than overlap.

A.2 Transforming from 3D coordinate system to 2D coordinate system

This techniques defines a plane in 2D from three points in 3D coordinate space. Points and lines are then transformed to the 2D coordinate system from the 3D system.

First the plane must be defined from three points. The coefficients of the plane A,B,C and D (see eqn A.1.1) must be found first. Two vectors, P and Q, are defined from the three points, and their cross-product is found. The result is a vector, R, normal to the plane. This vector is normalised to create a unit vector, r, by dividing each vector coefficient by the length of the vector, that is

$$length=(A^2+B^2+C^2)^{1/2} \qquad \text{(A.2.1)}$$

This is illustrated in Figure A.2.1

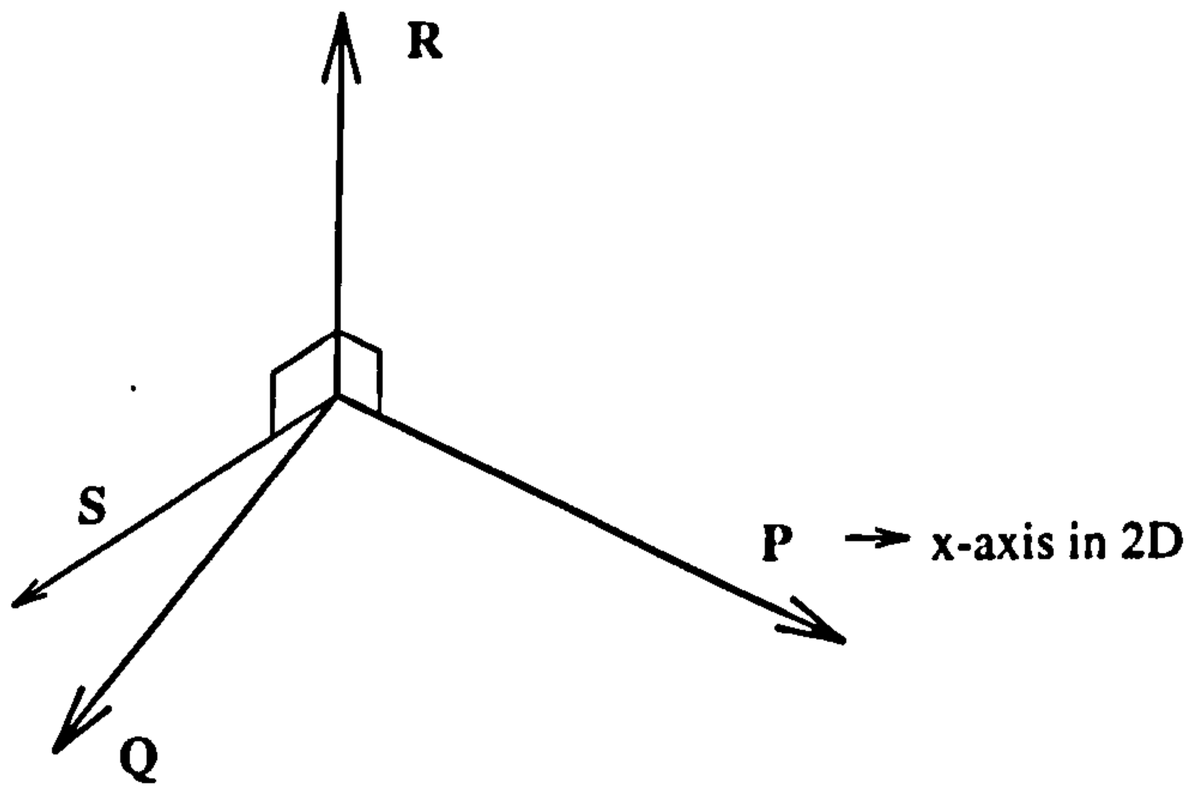


Figure A.2.1 Transforming to 2D coordinate system

The coefficients of this normal vector are also the A,B and C coefficients of the plane. Coefficient D is found by substituting one of the points which define the plane into equation A.1.1.

Next the 2D axes must be established. Vector P will form the x-axis in 2D, and one of its end-points will become the origin of this coordinate system. It then remains to define the y-axis. Vector r is normal to the 3D plane and so the cross-product of r and P is a vector, S, lying in the plane, normal to vector P. Vector S will therefore form the y-axis of the 2D coordinate system.

Points can now be transformed into the 2D system by finding the dot product of the points with each of the coordinate axes, P and S. The origin point of the 2D system will become coordinate (0,0) when its dot-product with each axis vector is found. Similarly, all of the other points will have new coordinates in this system. The lines joining them will be redefined according to the new 2D coordinate system, and so all calculations that require a 2D coordinate system can now be performed.

A.3 Intersection of two line segments

The intersection of two non-collinear line segments is a point determined by the following method. Two points lying on different vectors in 3D Cartesian space can be represented by

$or = oa + \lambda (ob - oa)$ where a,b are points and o is the origin

$os = oc + \gamma (od - oc)$ where c,d are points and o is the origin

Thus the intersection point, if it exists, occurs where or = os. This can be written

$$oa+\lambda(ob-oa)=oc+\gamma(od-oc) \qquad \text{(A.3.1)}$$

or in terms of x,y,z coordinates;

$$x_a-x_c=\gamma(x_d-x_c)-\lambda(x_b-x_a) \qquad \text{(A.3.2)}$$

$$y_a-y_c=\gamma(y_d-y_c)-\lambda(y_b-y_a) \qquad \text{(A.3.3)}$$

$$z_a-z_c=\gamma(z_d-z_c)-\lambda(z_b-z_a) \qquad \text{(A.3.4)}$$

It can be seen that there is an equation for each of the three coordinate axes, but only two unknowns, λ and γ. Therefore only two of the possible three equations need be used to find the unknowns, and only one unknown is required to substitute into the vector equation A.3.1. Problems arise, however, when the lines lie in planes parallel to the coordinate axes planes.

For example, if

$$x_b - x_a = 0 \tag{A.3.5}$$

then

$$\gamma = \frac{(x_a - x_c)}{(x_d - x_c)} \tag{A.3.6}$$

but if

$$x_d - x_a = 0 \tag{A.3.7}$$

that is, if the two lines lie in the yz plane, or a plane close to this, then the x coordinate equation cannot be used. By projecting both lines into a 2D plane, two equations and two unknowns exist with no degeneracies for two distinct lines.

The method used to projecting the lines into a 2D plane involved finding the normal to the plane described by the two lines. The cross product of the normal with the triangle side was found, and the unit vector of the result determined. This unit vector was then one of the axes of the projection plane, and the other axis was the triangle side itself. The other line was projected onto this plane by finding the dot product of the line's end points with both axes.

Where this method of finding the intersection of two lines was then used to determine whether a point lay inside or outside a triangle, the three triangle points and the test point were all projected into the same 2D plane initially, using just one os the triangle sides as one of the 2D plane axes. The mid points of the triangle sides were then calculated within this 2D projection plane.

A.4 Node position inside/outside of boundary faces

At various stages in the mesh linking program, it is necessary to determine whether a given node lies inside or outside of a closed boundary surface. A boundary surface will be closed if it consists of all the outer surface faces of a group of cells, which are found by the method described in Section 6.5. A list of nodes is then considered, checking each node in turn against each surface face in the following manner.

A distant point, well outside of the enclosed region is found. A straight line is created between this distant point and the node being tested. The intersection of this line is found with each face in the enclosing shell, as described in Section 6.5.

The sum of all the line-face intersections for that node is then found. If this sum is an odd number, then the node lies within the shell of faces. If the sum is an even number or zero, then the node lies outside of the shell. This is illustrated in Figure A.4.1 below.

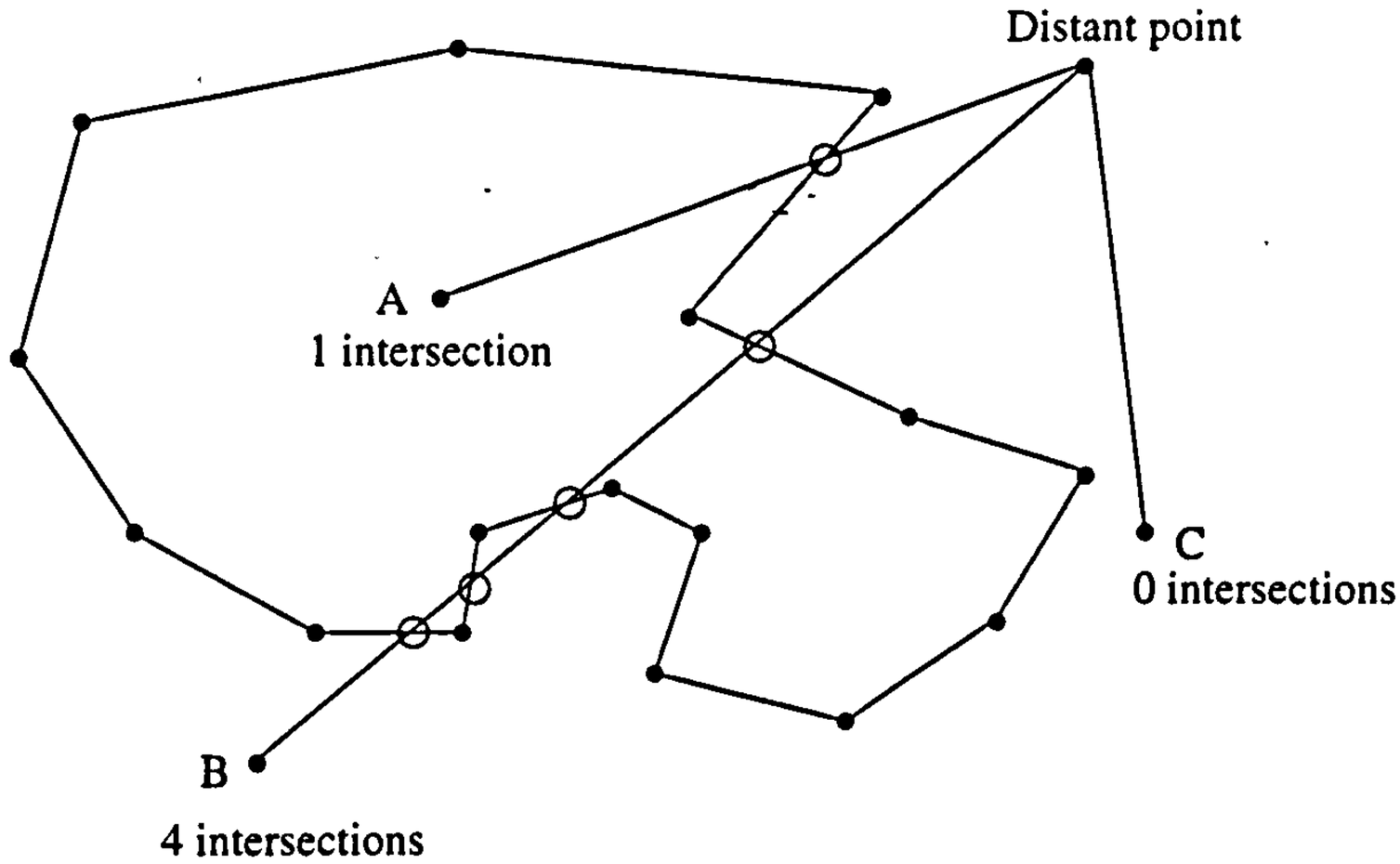


Figure A.4.1 Inside/outside test for nodes

In the above Figure, point A intersects the shell once and lies inside the shell, point B intersects the shell four times, an even number, and hence lies outside the shell, whereas point C has no intersections with the shell and so also lies outside of the shell.

REFERENCES

1. J.B. Heywood. Fluid motion within the cylinder of internal combustion engines - The 1986 Freeman scholar lecture. Journal of Fluids Engineering, 109, p.3-35. 1987

2. C. Arcoumanis, J.H. Whitelaw. Fluid mechanics of internal combustion engines - a review. Proc Instn Mech Engrs, 201 C1, p.57-74. 1987

3. D.R. Lancaster. Effects of engine variables on turbulence in a spark-ignition engine. SAE Paper 760159, p.671-688. 1976.

4. L.J. Kastner, T.J. Williams, J.B. White. Poppet inlet valve characteristics and their influence on the induction process. Proc Instn Mech Engrs, 178, Pt 1, 36, p.955-975. 1963-1964.

5. C. Vafidis. Influence of induction swirl and piston configuration on air flow in a four-stroke model engine. Proc Instn Mech Engrs, 198C, 8, p.71-79. 1984.

6. P. Steadman, J. Rooney. Principles of Computer-aided design. Pitman Publishing, 1987.

7. W. Brandstatter, R.J.R. Johns, G. Wigley. The effect of inlet port geometry on in-cylinder flow structure. SAE Paper 850499, 1986.

8. M. Namazian, S. Hansen, E. Lyford-Pike, J. Sanchez-Barsse, J. Heywood, J. Rife. Schlieren visualisation of the flow and density fields in the cylinder of a spark-ignition engine SAE Paper 800044, 1981.

9. K.H. Luo, H. Daneshyar. Measurement of valve flows of a four-valve S.I. engine as boundary conditions for in-cylinder flow models. SAE Paper 892097, 1989.

10. S. Bopp, F. Durst, C. Tropea. In cylinder velocity measurements with a mobile fibre optic LDA system. SAE Paper 900055, 1990.

11. R.S.W. Cheung, S. Nadarajah, M.J. Tindal, M. Yiannekis. An experimental study of velocity and Reynolds stress distributions in a production engine inlet port under steady flow conditions. SAE Paper 900058, 1990.

12. C. Vafidis, J.H. Whitelaw. Intake valve and in-cylinder flow development in a reciprocating model engine. Proc Instn Mech Engrs, 200 C2, 143-152. 1986.

13. C. Arcoumanis, Z. Hu, C. Vafidis, J.H. Whitelaw. Tumbling motion: a mechanism for turbulence enhancement in spark-ignition engines SAE Paper 900060, 1990.

14. S.H. El Tahry, B. Khalighi, W.R. Kuziak Jr. Unsteady-flow velocity measurements around an intake valve of a reciprocating engine. SAE Paper 870593, 1988.

15. B. Khalighi. Intake-generated swirl and tumble motions in a 4-valve engine with various intake configurations - flow visualisation and particle tracking velocimetry. SAE Paper 900059, 1990.

16. T. Wakisaka, Y. Hamamoto, S. Kinoshita. Turbulence characteristics in internal combustion engines. Bulletin of the JSME, 26, 212, p.254-261. 1983.

17. A. Murakami, M. Sakimoto, M. Arai, H. Hiroyasu. Measurement of turbulent flow in the combustion chamber of a D.I. Diesel engine. SAE Paper 900061, 1990.

18. N.W.H Armstrong and K.N.C Bray. Premixed turbulent combustion flowfield measurements using PIV and LST and their application to flamelet modelling of engine combustion. SAE Paper 922322, 1992.

19. A. Alkidas, P.V. Puzinauskas, R.C. Peterson. Combustion and heat transfer studies in a spark-ignited multi-valve optical engine. SAE Paper 900353, 1990.

20. D.L. Reuss, M. Bardsley, P.G. Felton, C.C. Landreth, R.J. Adrian. Velocity, vorticity and strain-rate ahead of a flame measured in an engine using particle image velocimetry. SAE Paper 900053, 1990.

21. M. Lorenz, K. Prescher. Cycle-resolved LDV measurements on a fired SI-Engine at high data rates using a conventional modular LDV system. SAE Paper 900054, 1990.

22. P.O Witze. A critical comparison of hot-wire anemometry and laser doppler velocimetry for I.C engine applications. SAE Paper 800132, 1980.

23. A. Ekchian, D.P Hoult. Flow visualisation study of the intake process of an internal

combustion engine. SAE Paper 790095, 1979.

24.N. Trigui, J.C.Kent, Y. Guezennec, W.-C. Choi. Characterisation of intake generated flow fields in IC engines using 3-D particle tracking velocimetry (3-D PTV). SAE Paper 940279, 1994.

25. H. Endres, H.-J. Neusser, R. Wurms. Influence of swirl and tumble on economy and emission of multi-valve SI engines. SAE Paper 920516, 1992.

26. S. Barraclough, Jaguar Cars Ltd. Private communication. 1994.

27. P. Girdinio, G. Molinari, S. Ridella. Trends in computer hardware and software for computer aided design applications. IEEE Transactions on Magnetics, 26, 2, p.761-766. 1990.

28. A.P. Watkins, C.J. Lea, M.Z. Gul. Development of advanced turbulence models for the calculation of flows in internal combustion engine cylinders. IMechE proceedings C430/025, 1991.

29. T. Kobayashi, K. Kitoh. A review of CFD methods and their application to automobile aerodynamics. SAE Paper 920338, 1992.

30. A.D. Gosman. Computer fluid dynamics applications to automobile design: progress and prospects. Use of supercomputers in the European automotive industry, Torino, April 1988.

31. C.T. Shaw. Using Computational Fluid Dynamics, Prentice Hall International, 1992.

32. B.E Launder, D.B Spalding. Mathematical models of turbulence. Academic Press, 1972.

33. S.V Patankar. Numerical heat transfer and fluid flow. Hemisphere, 1980.

34. A.D. Gosman. Multidimensional modelling of cold flows and turbulence in reciprocating engines. SAE Paper 850344, 1986.

35. T. Yamada, T. Inoue, A. Yoshimatsu, T. Hiramatsu, M. Konishi. In-cylinder gas motion of multi-valve engine - three dimensional numerical simulation. SAE Paper 860465, 1987.

36. A.D. Gosman, R.J.R. Johns. Development of a predictive tool for in-cylinder gas motion

in engines. SAE Paper 780315, 1978.

37. F.V. Bracco, F. Grasso. Sensitivity of chamber turbulence to intake flows in axisymmetric reciprocating engines. AIAA Journal, 21, p.607-640. 1983.

38. G.I. Ramos, W.A. Sirignano. Axisymmetric flow model with and without swirl in a piston-cylinder arrangement with idealised valve operation. SAE Paper 800284, 1980.

39. A.D. Gosman, Y.Y. Tsui, A.P. Watkins. Calculation of three dimensional air motion in model engines. SAE Paper 840229, 1985.

40. T. Wakisaka, Y. Shimamoto, Y. Isshiki, T. Shibata. Numerical simulation of gas flows in the cylinders of four-stroke cycle engines. Bulletin of JSME, 29, 258, p.4276-4284. 1986.

41. M. Kojima, H. Takata. Numerical analysis of flows in reciprocating engines. Bulletin of JSME, 29, 253, p.2183-2188. 1986.

42. S. Aita, A. Tabbal, G. Munck, K. Fujiwara, H. Hongoh, E. Tamura, S. Obana. Numerical simulation of port-valve-cylinder flow in reciprocating engines. SAE Paper 900820, 1990.

43. K. Naitoh, H. Fujii, T. Urushihara, Y. Takagi, K. Kuwahara. Numerical simulation of the detailed flow in engine ports and cylinders. SAE Paper 900256, 1990.

44. M.P. Errera. Numerical prediction of fluid motion in the induction system and the cylinder in reciprocating engines. SAE Paper 870594, 1988.

45. R. Taghavi, A. Dupont. Investigation of the effect of inlet port on the flow in a combustion chamber using multidimensional modelling. Transactions of the ASME, Journal of Engineering for Gas Turbines and Power, 111, p.479-484. 1989.

46. J.F. Le Coz, S. Henriot, P. Pinchon. An Experimental and computational analysis of the flow field in a four-valve spark ignition engine-focus on cycle-resolved turbulence. SAE Paper 900056, 1990.

47. A.A. Amsden, P.J. O'Rourke, T.D. Butler, K. Meintjes, T.D. Fansler. Comparison of computed and measured three-dimensional velocity fields in a motored two-stroke engine. SAE Paper 920418, 1992.

48. A.D. Gosman, A.M.Y. Ahmed, Measurement and multidimensional prediction of flow in axisymmetric port/valve assembly. SAE Paper 870592, 1988.

49. S. MacDonald, Adapco Ltd. Private communication. 1990.

50. I-DEAS Manuals Version VI. Structural Research and Dynamics Corporation Ltd. 1993.

51. S.A Coons. Surfaces for computer aided design of space forms. Project MAC Report TR-MAC-41, Mass. Institute of Technology, 1967.

52. K.H. Luo and K.N.C. Bray, 3D Simulation of induction port flow of a four-valve engine configuration. SAE Paper 920586, 1992.

53. K.H. Luo and K.N.C. Bray. Full-Field Simulation of Engine Induction Port Flow using Second-Moment Closure Turbulence Model, CFD News, 2, 2, Institute of Mathematics. 1990.

54. H.H Dannelongue, P.A Tanguy. Three-dimensional adaptive finite-element computations and applications to non-Newtonian fluids. International

55. F. Moukalled, S. Acharya. A local adaptive grid procedure for incompressible flows with multigridding and equidistribution concepts. International Journal for Numerical Methods in Fluids, 13, 9, p.1085-1111. 1991.

56. A. Evans, M.J. Marchant, N.P. Weatherill, J. Smeltzer, D.K Natakusumah. A point enrichment strategy for polygonal finite volume meshes. Appl. Math. Modelling, Vol. 16, p.562-575. November 1992.

57. RAMPANT Manuals V3.0. Fluent Inc. 1994.

58. STAR-CD Version 2.1 Manuals, Computational Dynamics Limited, 1991.

59. R.I Issa. Solution of the implicitly discretised fluid flow equations by operator splitting. J. Comp. Physics, 62, p.40-65, 1986.

60. Computational Dynamics. Private communication. 1991.

61. FIDAP Manuals Rev. 6.0. Fluid Dynamics International, 1991.

62. J.N Reddy. An introduction to the finite element method. McGraw-Hill. 1984

63. P.L. George. Automatic Mesh Generation. John Wiley & Sons, 1991.

64. W.A. Cook. Body oriented (natural) coordinates for generating three dimensional meshes. Int Journ Num Meth in Eng, Vol 8, p.27-43. 1974.

65. W.J. Gordan, C.A. Hall. Construction of curvilinear coordinate systems and applications to mesh generation. Int Journ Num Meth in Eng, Vol 7, p.461-477. 1973

66. O.C. Zienkiewicz, D.V. Phillips. An automatic mesh generation scheme for plane and curved surfaces by 'isoparametric' co-ordinates. Int Journ Num Meth in Eng, Vol 3, p.519-528. 1971.

67. M.S. Shephard, P.M. Finnigan. Toward automatic mesh generation, State-of-the-art surveys on Computational Mechanics. Eds A.K. Noor, J.T. Oden, Chapter 11 p.335-366.

68. Y.A. Yerry, M.S. Shephard. Automatic three-dimensional mesh generation by the modified-octree technique. Int Journ Num Meth in Eng, Vol 20, p.1965-1990. 1984.

69. J. Peraire, J. Peiro, L. Formaggia, K. Morgan, O.C. Zienkiewicz. Finite element Euler computations in three dimensions. Int Journ Num Meth in Eng, Vol 26, p.2135-2159. 1988.

70. T.J. Baker. Unstructured meshes and surface fidelity for complex shapes. AIAA-91-1591-CP, p.174-728. 1991.

71. C.M. Maksymiuk, M.L. Merriam. Surface reconstruction from scattered data through pruning of unstructured grids. AIAA-91-1584-CP, p.648-653. YEAR??

72. P.L. George, F. Hecht, E. Saltel. Automatic 3D mesh generation with prescribed meshed boundaries. IEEE Transactions on Magnetics, 26, 2, p.771-774. 1990.

73. C.G. Armstrong, T.K.H. Tam, D.J. Robinson, R.M. McKeag, M.A. Price. Automatic generation of well structured meshes using medial axis and surface subdivision. Advances in design automation, DE-Vol 32-2, Vol-2 ASME. 1991.

74. H. Jin, R.I. Tanner. Generation of unstructured tetrahedral meshes by advancing front

technique. Int Journ Num Meth in Eng Vol 36 p.1805-1823. 1993.

75. J.C. Cavendish, D.A. Field, W.H. Frey. An approach to automatic three-dimensional finite element mesh generation. Int Journ Num Meth in Eng Vol 21 p.329-347. 1985

76. A.J. Bocci. Recent developments in CFD at ARA. Aeronautical Journal, p.111-123. April 1991.

77. N.P. Weatherill. On the combination of structured-unstructured meshes. Numerical grid generation in computational fluid mechanics '88. Eds. S. Sengupta, J. Hauser, P.R. Eiseman, J.F. Thompson, Pineridge Press, 1988.

78. S. Aita, N. Montmayeur, J.F. Levy, A.Tabbal, G. Munck. Computational fluid dynamics for industrial development and design. International IBM workshop on Industrial applications of CFD, Italy, 16-18 October, 1990.

79. J.A. Shaw, J.M. Georgala, A.J. Peace, P.N. Childs. The construction, application and interpretation of three-dimensional hybrid meshes. Third International Conference on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, Barcelona, Spain, 3-7 June, 1991.

80. Fluent-unstructured, Fluent Inc. 1994.

81. K. Hanna, Fluent Inc. Private communication, 1993.

82. C. Smith. Solid geometry. Macmillan and co. Ltd, 1931.