FINITE ELEMENT ANALYSIS OF COMPRESSIBLE FLOWS

Luke Timothy Felthun

A dissertation submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science in Engineering.

Johannesburg, 1995

DECLARATION

I declare that this dissertation is my own, unaided work. It is being submitted for the Degree of Master of Science in Engineering in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other university.

L Felth

16 th day of February 1995

ABSTRACT

In this research a finite element analysis program was developed for the modelling of general compressible Euler flows. An explicit Taylor-Galerkin algorithm was used as the flow solver and was used in conjunction with a flux-corrected transport algorithm in order to obtain high shock resolution without numerical oscillations and overshoots. The solver was applied to two and three dimensional geometries. An axisymmetric extension of the Taylor Galerkin algorithm was also developed.

For the two dimensional code, a fully automatic mesh generator was implemented which was able to generate meshes for completely arbitrary geometries, as well as an adaptive refinement algorithm which performs an error analysis on the solution and refines and coarsens the mesh appropriately in order to maintain an optimal mesh resolution. The automatic mesh generator dramatically reduced problem setup time and the adaptive refinement algorithm reduced computer time by up to 90%.

A number of test cases were performed covering a wide range of compressible flows including steady and unsteady flows in air, using the ideal gas model, and shocks in liquids, using the Tait model. Within the limitations of the inviscid and real gas assumptions made, accurate results were obtained.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

B	Empirical constant for Tait equation of state
C_e	FCT limiter
c	Speed of sound
c_d	Diffusion coefficient
e	Specific energy
F_i	Flux vector in direction i
M	Mach number
M_C	Consistent mass matrix
M_L	Lumped mass matrix
N_i	Shape function for node i
n	Empirical constant for Tait equation of state
	Normal vector
p	Pressure
p'	Modified pressure
R	Ideal gas constant
S	Source term for axisymmetric Euler equation
T	Temperature
t	Time
	Tangent vector
U	Vector of conservation variables
U^n	Solution at time n
U^l	Low order solution
U^h	High order solution
u	Velocity in x direction
u_i	Velocity in coordinate direcion i
v	Velocity in y direction
w	Velocity in z direction
x_i	Coordinate value in direction i

Greek Symbols

β	Angle of reflection for steady flow
γ	Ratio of specific heats
δ	Kronecker delta
θ	Wedge angle
ρ	Density
χ	Triple point trajectory angle
Ψ	Angle of reflection

1. INTRODUCTION

The purpose of this research was to develop a computational fluid dynamics (cfd) program for the modelling of compressible fluid flows. Incompressible flows are not considered in this work as fundamentally different approaches are generally taken in their analysis.

The major criteria of this research was that of generality. The program to be developed had to be a general purpose tool that could be used to model an as wide range of compressible flow problems as possible. It was to be applied to both steady flow applications, as typical in high speed aerodynamics, and unsteady flows, as typical in explosive phenomena. All types of geometries, two dimensional, three dimensional and axisymmetric, were to be modelled. One of the most important criteria was that the code had to easily be able to handle completely arbitrary flow geometries, possibly with curved and interior boundaries, that is with holes in the domain. This criteria arose as many existing codes have difficulty handling complex geometries due to the fact that these schemes are based on structured grids which do not readily accommodate arbitrarily complex flow domains.

A fairly unique application that was also of interest was that of modelling shock wave phenomena in liquids[1-2]. Liquids are usually modelled as being incompressible but liquid shock wave phenomena are the result of the compressibility effects of the liquid and hence in this case have to be taken into account. The program therefore needed to be able to model different mediums, requiring the use of different equations of state. Typically gases are modelled using an ideal gas equation of state and compressible liquids are modelled using the Tait equation of state[2].

As it would not be practical in the context of this work to develop a code that could handle every possible compressible flow situation, certain limitations were made. The most fundamental assumption made was that viscous effects would be neglected, that is, the flow is inviscid. This is a widely used assumption when modelling compressible flows mainly because viscous effects tend to play a less dominant role than in incompressible flows. This is especially the case in highly transient problems where boundary layers and viscous vortices do not have sufficient time to develop. Another simplification made for modelling air flows is the assumption of ideal gas behaviour. Gases at higher pressures and temperatures tend to deviate from ideal gas behaviour, at which point some form of real gas model would be more appropriate.

The equations that describe the above inviscid compressible flow problems and hence for this work need to be solved, are the compressible Euler equations.

1.1 Numerical Solution Schemes

There are a wide variety of schemes available for solving the compressible Euler equations. A brief review of some of the families of techniques is given below.

Finite difference methods were at one time the most widely used schemes and due to their simplicity are still popular. These are structured schemes based on an orthogonal grid of points. Typically, central difference approximations are used to approximate the space derivatives and a Taylor series used to approximate the time derivatives. Schemes include those of MacCormack[3], which was the standard scheme during the 1970s, and the more modern implicit scheme of Beam and Warming[4]. Finite difference methods, primarily due to their orthogonal grid structure, are relatively simple to implement and generally computationally efficient. Their main limitation, which is also due to their orthogonal grid structure, is their inflexibility in accommodating complex geometries.

Finite volume schemes[5] are unstructured schemes which are based on a mesh consisting of triangular or quadrilateral cells in two dimensions and tetrahedral or hexahedral cells in three dimensions. With this approach the conservation laws in integral formulation, that is, in the form of the Reynold's transport theorem, are discretised directly into physical space. These schemes to some extent involve a more physical interpretation of the compressible Euler equations as opposed to a purely mathematical approach. Typically the flow variables are approximated within each cell by a constant or linear discontinuous interpolation while fluxes are evaluated on the boundaries of the cells.

Spectral methods[6] are highly regarded for their ability to resolve fine flow features and are frequently applied to incompressible flow problems involving flow instability and turbulence. These are high order methods which usually use a Fourier trigonometric or Chebyshev polynomial series to represent the solution. Their application to compressible flow problems have been limited primarily due to their poor resolution of discontinuities. Improving this is a current field of research[7] but these techniques have not yet reached the stage of development where they could be considered as a practical method for solving general compressible flow problems.

Finite element schemes[8] are unstructured schemes originally developed for structural analysis and later applied to general field problems. Essentially the domain is divided into discrete elements, as in the finite volume case, with the flow variables being approximated by an interpolation polynomial within each element, typicaliy for compressible flow calculations a linear interpolation is used, though for other types of problems higher order interpolations are normal. The differential equations are then cast into a weighted residual or variational form. Major advantages of finite element schemes are their ability to accommodate complex geometries and their natural handling of boundary conditions, both of which are frequently problems in other approaches. The finite element approach has to a large extent become the de facto standard for modelling problems in structural mechanics and is becoming increasingly popular in fluid mechanics.

Structured schemes can be applied to complex geometries through various methods. One technique is to cover the whole region with a grid and then to delete sections of the grid which fall outside the flow domain. Difficulties then arise with cells that lie on the boundary and must then be only partially deleted[9]. Another approach is to subdivide the domain into blocks, possibly curved and distorted if necessary, and then mapping the structured grid into each block. This is typically referred to as domain decomposition or block structuring[10]. Examples of domain decomposition schemes include spectral element schemes[11], which are based on the spectral methods described above, and chimera overset schemes[12] where different blocks, instead of having to be adjacent to each other, are allowed to overlap. A third approach is to derive the scheme in terms of generalised orthogonal coordinates[13] which are more readily adapted to curved boundaries than rectilinear coordinates.

While it is possible to apply structured schemes to model arbitrarily complex geometries, using these schemes in this context can be complicated and tedious. Further development in structured schemes are necessary if they are to become widely used for complex geometry problems. In order to avoid these difficulties it was decided to use an unstructured scheme as the basis for this program. Both finite volume and finite element schemes, both being unstructured schemes, were therefore considered as possible candidates for implementation. Both schemes appear equally competent in modelling compressible flows and both approaches could have equally well fulfilled the criteria of this research. The finite element method tends to be a more mathematical approach whereas the finite volume method could be regarded as a more engineering type approach. The finite element scheme was chosen but this would have to be regarded largely as a decision based on personal preference rather than any strictly objective criterion.

There are three main families of finite element schemes that have been applied to compressible flows, least squares[14], Streamline Upwind Petrov Galerkin (SUPG)[15-16] and Taylor-Galerkin[17-21] schemes. While all of these are applicable to both transient and steady flows there has been very little research on the application of least squares and SUPG methods to transient flows. The Taylor-Galerkin method has been applied extensively to both steady and unsteady flows and it was on this basis that the it was chosen. An existing axisymmetric version of the Taylor-Galerkin algorithm was not found so an axisymmetric extension of this algorithm is derived in this work.

One of the biggest problems encountered in virtually all numerical schemes when modelling compressible flows, is the existence of numerical instabilities and overshoots in the presence of discontinuities, for example shock waves and contact discontinuities, in the solution which eventually cause the entire solution to become unstable. These instabilities are typically stabilised by adding artificial viscosity or diffusion to the solution, resulting in the discontinuity being smeared over several elements. In order to maintain stability while maintaining high shock resolution the Taylor Galerkin algorithm is used in conjunction with a flux-corrected transport (FCT) algorithm[22-28].

Another choice that needed to be made was whether to use either an explicit or implicit implementation of the algorithm. For a single time step, explicit implementations are by far more efficient but are subject to a very stringent stability criterion which limits the magnitude of the time step that can be used. This criterion is based on limiting the distance any wave can propagate to some proportion of the smallest mesh interval, essentially a wave must not be able to jump over any element during a time step. Implicit implementations require the solution of a matrix system of equations and are therefore computationally more expensive, but the restriction on the size of the timestep is removed. Certainly in highly transient problems, due to the rapid change in the solution with respect to time, in order to maintain time accuracy a very small time step would have to be used anyway, so the stability criterion is not a serious restriction. When using the algorithm to time step the solution to steady state the restriction is more limiting but not seriously so as high speed flows tend to converge rapidly to steady state. The areas where explicit algorithms would be seriously limiting would be in the handling of viscous flows where viscous boundary layers and vortices take some time to develop and a stability criterion based on wave propagation speeds would be totally inappropriate. Implicit implementations of the Taylor-Galerkin algorithm do exist[21], but for the Euler flows that were to be modelled an explicit algorithm was considered to be more appropriate.

1.2 Mesh Generation

All numerical schemes require some form of grid or mesh to cover the geometry. The Taylor-Galerkin scheme implemented uses a triangular mesh in two dimensions and a tetrahedral mesh in three dimensions. This is not a fundamental requirement as it is possible to use quadrilateral and hexahedral elements. Manual and semi-automatic methods of generation of the finite element mesh is time consuming and tedious creating difficulties similar to that encountered when using structured schemes. The most common method of mesh generation is the use of parametric mapping techniques[29-30]. This requires the user to divide the flow domain into arbitrarily shaped quadrilateral (two dimensional) or hexahedral (three dimensional) blocks which the mesh generator then subdivides into finite elements. However if this approach is used, not only is it time consuming for the analyst setting up the problem to subdivide the domain into blocks, but then the justification of using an unstructured scheme is lost as a structured mesh could then just as well be used within each of the quadrilateral/hexahedron blocks.

It would however be desirable for the analyst just to define the boundaries of the domain and for the mesh generator to automatically generate the triangular mesh. Two approaches used to do this are Delauney triangulation[31] and the advancing front method[32-33] which have both been applied to the generation of triangular and tetrahedral meshes. Delauney triangulation requires an existing set of nodal coordinates which it then triangulates. The advancing front technique generates nodes and elements simultaneously which eliminates the need for a separate algorithm for the creation of nodal points. The advancing front algorithm has also been widely used for the creating meshes for compressible flow problems and to the Taylor-Galerkin method in particular and was chosen on this basis. This does not mean to imply however that Delauney triangulation is inappropriate for these type of problems.

Automatic mesh generation was only applied here to the generation of two dimensional triangular meshes. Although three dimensional versions of these mesh generators do exist[20, 31] which generate tetrahedral elements, they were not implemented in this work. Instead the parametric mapping approach described in Zienkiewicz[30-31] was used to generate the three dimensional tetrahedral meshes. This was chosen purely on the basis that it is easier to implement as it does not appear to have any other advantages.

1.3 Solution Adaptive Meshes

If a computationally efficient algorithm is wanted, having optimal mesh resolution becomes necessary. The mesh needs to be finer in areas of rapid changes in flow variables and ideally coarser elsewhere. If the mesh is not fine enough in areas with features with steep gradients, these features will not be adequately resolved. If the mesh is too fine in areas where the flow is smooth, that is where gradients are small, a lot of computational effort is wasted on the excess elements. In general for these types of problems it is impractical for the analyst to try to create and maintain an optimal mesh. In addition, with transient problems where the solution changes continually no single optimal mesh exists. Typically the usual approach is to use a uniformly fine mesh over the whole domain, in an attempt to obtain the desired accuracy, despite the fact that this is computationally expensive.

The general approach behind solution adaptive meshes is for the computer to determine the optimal mesh resolution and to automatically adapt the mesh accordingly every few time steps. The optimal resolution is determined from an error analysis that is performed on the solution at the current time step. There are two main approaches of adapting the mesh. One is to use the automatic mesh generator which created the initial mesh to remesh the domain[20, 33-34], with the size of each element created being calculated based on the error on the previous mesh. The other approach is the adaptive refinement[35] approach which refines the mesh by subdividing existing elements into smaller elements and possibly coarsening it again at a later stage. A third possible, though not widely used, approach, is the r refinement method[36] where a mesh is stretched and distorted, without increasing the number of elements or nodes or the way they are connected, in an attempt to create an optimal mesh distribution. This approach however has not proven to be very flexible.

The computational expense in regenerating the entire mesh is considerably higher than subdividing elements in the initial mesh into smaller elements. Though implementations of automatic mesh regeneration have been developed specifically for transient problems[34] where only the appropriate sections of the mesh are regenerated, it has problems with accurately interpolating the solution from the old to the new mesh. Automatic mesh regeneration however has been very successfully used in steady flow problems when the mesh is only regenerated a few times and as only the converged solution is of interest, loss of accuracy in interpolating the solution between the old and the new meshes is not serious. Adaptive refinement techniques, however, are fast and have been demonstrated to work well with both transient and steady flows. The adaptive refinement algorithm of Löhner[35]

was chosen having successfully been applied to similar flow problems to those which were of interest here.

Using an automatic mesh generator in conjunction with some form of automatically adaptive mesh has the advantage of reducing the analysts role in the solution process. Ideally, the analyst should only need to supply the minimum information necessary to define the problem, with the computer handling the meshing and solution details. This approach, essentially treating the solver as a 'black box', into which a problem is put and a solution obtained without further intervention by the analyst, is taken with the two dimensional code implemented in this work.

1.4 Thesis outline

The topics in this thesis are dealt with in the same general order that they are treated in this introduction. In chapter 2 the Taylor-Galerkin algorithm is described and its extension to axisymmetric flows and liquid shocks derived. The advancing front automatic mesh generator is described in chapter 3 and the adaptive refinement algorithm in chapter 4. Algorithm logic is emphasized and programming details minimised in these chapters. Comparisons of numerical results with analytical and experimental results are performed in chapter 5 where the application of the finite element code to general flow problems is also demonstrated. In Appendix A, all the finite element equations used in the program are integrated and expanded into full matrix form. In appendix B, auxiliary algorithms such as contour plotting and three dimensional viewing are described.

2. FINITE ELEMENT SOLVER

2.1 Introduction

The finite element solver used in this work is an explicit Taylor-Galerkin procedure originally developed for convective transport problems by Donea[17] and extended to non-linear hyperbolic equation systems by Löhner, Morgan and Zienkiewicz[18] who later applied it specifically to compressible Euler flows[19]. A slightly modified version of the Taylor-Galerkin algorithm was applied to three dimensional compressible Euler problems in Peraire et al[20], which is the implementation used here for both the two and three dimensional problems. An implicit version of the Taylor-Galerkin algorithm is described in Hassan et al[21].

The basic approach of the Taylor-Galerkin algorithm is to discretise the hyperbolic equation system in space using a Galerkin weighted residual procedure and to discretise the equations in time using a second order Taylor series. In the implementation of Peraire et al[19] used here, the Taylor series is replaced by a central difference approximation, however the effective change in the algorithm is minimal and it is still regarded as a member of the Taylor-Galerkin family of algorithms.

In this chapter the Taylor-Galerkin procedure is derived and extended to deal with axisymmetric flows. The use of the procedure in conjunction with the Tait equation of state is also described. The Taylor-Galerkin scheme, like all high order schemes, will experience Gibbs like oscillations and overshoots in the vicinity of discontinuities. It is therefore used in conjunction with a flux-corrected transport (FCT) algorithm[22-28] which is used to stabilise the solution in the vicinity of shocks while still obtaining high shock resolution.

In appendix A all the finite element equations in this chapter are integrated and expanded into full matrix form.

2.2 The Compressible Euler Equations

The compressible Euler equations for inviscid compressible flow in three dimensions, a hyperbolic system of equations, can be expressed in conservation form, using the summation convention, as follows

$$\frac{\partial U}{\partial t} + \frac{\partial F_i}{\partial x_i} = 0 \qquad i=1,\ 2,\ 3 \tag{1}$$

where x_i is the coordinate direction and

$$U = \begin{bmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ \rho u_3 \\ \rho e \end{bmatrix}, \qquad F_i = \begin{bmatrix} \rho u_i \\ \rho u_1 u_i + p\delta_{1i} \\ \rho u_2 u_i + p\delta_{2i} \\ \rho u_3 u_i + p\delta_{3i} \\ u_i(\rho e + p) \end{bmatrix} \tag{2}$$

where δ is the Kronecker delta, u_1 , u_2 and u_3 are the fluid velocities in each coordinate direction, p is the pressure, ρ the density and e the specific internal energy. The first row in the above system of equations is the conservation of mass equation, the next three rows are the conservation of momentum equations for each coordinate direction, and the last row is the conservation of energy equation. U is the vector of conservation variables for which the system is solved. The two dimensional Euler equations are obtained simply by leaving out the third momentum equation and summing i from 1 to 2.

The above set of equations is completed by an equation of state, from which the pressure (p) in the conservation of momentum and energy equations is calculated.

2.2.1 Equations of state

For compressible flows in air, the isentropic equation of state for a ideal gas is

$$\frac{p_2}{p_1} = \left(\frac{\rho_2}{\rho_1}\right)^{\gamma} \tag{3}$$

where γ is the ratio of specific heats (γ = 1.4 for air). The ideal gas equation of state can also be expressed as

$$p = \rho RT \tag{4}$$

which for use with the compressible Euler equations in the above form is more appropriately expressed in terms of the conservation variables as follows

$$p = (\gamma - 1)\rho(e - \frac{1}{2}u_i u_i) \tag{5}$$

The isentropic equation of state for liquids can be written in the following form known as the Tait equation of state[1-2].

$$\frac{p_2 + B}{p_1 + B} = \left(\frac{\rho_2}{\rho_1}\right)^{\gamma} \tag{6}$$

where B and γ are empirically determined constants. It must be noted that γ is no longer the ratio of specific heats as in (3). It can be seen that the Tait equation of state (6) is obtained from the perfect gas equation of state (3) by replacing p by p' defined below

$$p' = p + B \tag{7}$$

In essence the Tait equation is obtained by adding an offset pressure (B) to the ideal gas equations. The Tait equation is therefore assuming that a liquid behaves like an ideal gas at very high pressure. This substitution can be made in any equations derived using the perfect gas equation of state to convert them to Tait equation models.

Equation (5) then becomes

$$p' = p + B = (\gamma - 1)\rho(e - \frac{1}{2}u_i u_i) \tag{8}$$

By using the equation of state in the above form, all the substitutions for pressure are automatically performed in the Euler equations. It is therefore possible to perform simulations using the Tait equation without making any modification to any of the equations or the solver as long as the initial conditions are calculated appropriately.

2.3 Finite Element Discretisation

The finite element discretisation of the above system will now be described. Equation (1) is cast in a Galerkin weighted residual form[37]

$$\int_{\Omega}\left(\frac{\partial U}{\partial t} + \frac{\partial F_i}{\partial x_i}\right) N_j \, d\Omega = 0 \tag{9}$$

where the integrations are performed over the element area and N_j are the finite element shape functions, which in Galerkin formulations are also used as the weighting functions. Applying the Gauss divergence theorem to (9) :

$$\int_{\Omega}\frac{\partial U}{\partial t} N_j \, d\Omega = \int_{\Omega} F_i \frac{\partial N_j}{\partial x_i} \, d\Omega - \int_{\Gamma} F_i n_i N_j d\Gamma \tag{10}$$

where the integrations over Γ are performed over the boundary and n_i is the unit outward normal to the boundary. From here on, the following finite element interpolations are used for U and F_i

$$U = U_k N_k , \qquad F_i = F_i(U_k) N_k \tag{11}$$

Substituting these into (10) :

$$M_{jk}\frac{dU_k}{dt} = \int_{\Omega} F_i \frac{\partial N_j}{\partial x_i} d\Omega - \int_{\Gamma} F_i n_i N_j d\Gamma \tag{12}$$

where M_{jk} , the consistent mass matrix, is defined as

$$M_{jk} = \int_{\Omega} N_j N_k d\Omega \tag{13}$$

A central difference approximation is then applied to the time derivative. In previous versions of this algorithm[17-19], a second order Taylor expansion was used instead.

$$M_{jk}\frac{U_k^{n+1} - U_k^n}{\Delta t} = \int_\Omega F_i^{n+\frac{1}{2}} \frac{\partial N_j}{\partial x_i} d\Omega - \int_\Gamma F_i^{n+\frac{1}{2}} n_i N_j d\Gamma \tag{14}$$

The consequence of this central difference approximation is that the fluxes now need to be evaluated at time n+1/2. For this to be done, an approximation to the solution at time n+1/2 needs to be evaluated. The following approximation to δU, the half timestep solution, is obtained by substituting the partial derivatives of U in equation (1) with differentials.

$$\delta U_e = \frac{-\Delta t}{2} \frac{\partial F_i}{\partial x_i} \tag{15}$$

where δU_e is constant across each element and discontinuous between elements. A piecewise linear discontinuous approximation to $U^{n+1/2}$ is then obtained as follows

$$U^{n+\frac{1}{2}} = U^n + \delta U_e \tag{16}$$

The values for the fluxes to be substituted into equation (14) are calculated using the above value:

$$F_i^{n+\frac{1}{2}} = F_i(U^{n+\frac{1}{2}}) N_k \tag{17}$$

This implementation of the Taylor-Galerkin procedure is referred to as a two step algorithm, since the solution is obtained in two main steps, first an approximation of the solution at time n+1/2 and then the evaluation of equation (14) which will give the solution at time n+1.

Once all the integrations have been performed, equation (14) can be assembled in the following matrix form :

$$M_{jk}\Delta U_k = RHS_j \tag{18}$$

This being a fully explicit algorithm, this system is never assembled to create a full matrix equation system but is solved explicitly and iteratively as follows

$$(M_L)_j \Delta U_j^r = \left(\sum_e (RHS_e - (M_C - M_L)_e \Delta U_j^{r-1})\right)_j \qquad (19)$$

where r is the iteration number with $\mathbf{U}^0 = \mathbf{0}$, subscript j represents assembled nodal quantities and subscript e represents unassembled element quantities, the so called element contributions which are referred to later in the flux-corrected transport algorithm. M_L is the lumped mass matrix which is obtained by summing up each row of the consistent mass matrix and assigning this value to the diagonal. Essentially the element contributions are calculated on an element by element basis (represented by the right hand side of the above equation) and then assembled into the solution vector. This process is repeated for each iteration. Three iterations are performed when time accurate solutions are needed for transient problems but when only the steady state solution is of interest, only one iteration needs to be performed.

2.4 Axisymmetric Implementation

In cases where both the domain geometry and the flow conditions are axisymmetric it is convenient to represent this essentially three dimensional situation in two dimensions by only modelling a plane through the axis of symmetry. It must be emphasized that the axisymmetric assumption can only be used when the flow is also axisymmetric as an axisymmetric geometry can have non axisymmetric flow conditions, for example, a cone inclined at an angle to the flow. It is then necessary to add a source term to the Euler equations to compensate for three dimensional effects, essentially as the fluid flows outward axially the expansion of the flow must be taken into account. An axisymmetric implementation of the Taylor-Galerkin algorithm was not found, but fortunately it is easy to modify it to accommodate extra terms in the hyperbolic equation system. The axisymmetric[1] form of equation (1) is written

$$\frac{\partial U}{\partial t} + \frac{\partial F_i}{\partial x_i} = S \qquad (20)$$

where

$$S = -\frac{1}{r}\begin{bmatrix} \rho v \\ \rho v u \\ \rho v^2 \\ v(\rho e + p) \end{bmatrix} \tag{21}$$

In the above essentially 2D set of equations the x axis is the axis of symmetry and the y axis is used as the radial axis and v is the fluid velocity in the radial direction. As can be seen above, the source term is only non zero when the radial velocity is non zero. The source term is also inversely proportional to the distance from the radius. The finite element formulation follows similarly to before. Applying the Galerkin weighted residual formulation to (20) gives

$$\int_\Omega \left(\frac{\partial U}{\partial t} + \frac{\partial F_i}{\partial x_i} - S \right) N_j \, d\Omega = 0 \tag{22}$$

Applying the Gauss divergence theorem gives

$$\int_\Omega \frac{\partial u}{\partial t} N_j \, d\Omega = \int_\Omega F_i \frac{\partial N_j}{\partial x_i} \, d\Omega - \int_\Gamma F_i n_i N_j d\Gamma + \int_\Omega S_i N_j d\Omega \tag{23}$$

The derivations follow similarly to those in section 2.3, the axisymmetric form of equation (14) then being

$$M_{jk} \frac{U_k^{n+1} - U_k^n}{\Delta t} = \int_\Omega F_i^{n+\frac{1}{2}} \frac{\partial N_j}{\partial x_i} d\Omega - \int_\Gamma F_i^{n+\frac{1}{2}} n_i N_j d\Gamma + \int_\Omega S_i^{n+\frac{1}{2}} N_j d\Omega \tag{24}$$

The axisymmetric version of (15), the first step of the algorithm, is then

$$\delta U_e = -\frac{1}{2}\Delta t \left(\frac{\partial F_i}{\partial x_i} - S \right) \tag{25}$$

The rest of the algorithm proceeds as before to the two dimensional implementation. Using the above formulation, axisymmetric geometries can be treated as two dimensional geometries with the x axis being used as the axis of symmetry and the y axis as the radial direction.

2.5 Stability Criterion

This algorithm, being fully explicit, will be subject to a Courant type stability criterion[20] which limits the allowable timestep, preventing the solution from becoming unstable. This stability criterion effectively limits the distance over which a wave can propagate during any time step. The following equation must be satisfied for every element for stability :

$$\Delta t \leq \frac{\beta h_e}{\sqrt{3}(\sqrt{u_i u_i}+c)} \tag{26}$$

where h_e is the smallest side length of an element and β is a safety factor which is normally set to 0.4 for transient problems and 0.9 for steady state problems. It is possible, particularly during transient problems for flow conditions to change resulting in changes in the stability of the solution. It is therefore appropriate to regularly calculate the timestep throughout the simulation. This is usually done every five timesteps after the mesh has been adapted.

2.6 Boundary conditions

In this algorithm the boundary conditions are applied in a weak form through the boundary integral of (14). The effect of the boundary integral is to con... in the flow variables on the boundary to the values specified by $U^{n+1/2}$ calculated in the first step. The value of $U^{n+1/2}$ used in the integral is corrected for solid boundaries by applying a linearised characteristics analysis[38].

At a solid wall the normal velocity is prescribed to be zero. The characteristics analysis provides the following predictions for pressure and density at a solid wall :

$$\begin{aligned} v_{n_c} &= 0 \\ v_{t_c} &= v_{t_p} \\ p_c &= p_p + v_{n_p}\rho c \\ \rho_c &= \rho_p + v_{n_p}\frac{\rho}{c} \end{aligned} \tag{27}$$

Subscripts n and t represent normal and tangential directions respectively and subscripts p and c represent predicted and corrected values respectively. For free boundaries used for inflow and outflow, no modification is made to the value obtained for $U^{n+1/2}$ from (16). If it is desired for a boundary to be constrained, it is only necessary to assign the appropriate values for $U^{n+1/2}$. Note that the modified values of $U^{n+1/2}$ are only used in the boundary integral.

2.7 Flux-corrected Transport (FCT)

The Taylor-Galerkin algorithm, like most numerical schemes, has difficulty in the vicinity of discontinuities in the solution. Typically overshoots and oscillations will develop at the discontinuities eventually causing the solution to become unstable and break down. The traditional approach of dealing with this is to add diffusion or artificial viscosity to stabilise the solution. This is effective but has the effect of smearing the discontinuity over about 6 to 9 elements. One approach of stabilising the solution while still maintaining high shock resolution is the flux-corrected transport (FCT) algorithm originally developed by Boris and Book[22-24] and multi-dimensionalised by Zalesak[25]. It was applied to finite element computations in Erlebacher[26] and Parrot and Christie[27]. It was further developed by Löhner et al[28] whose version is applied here.

The principle behind the flux-corrected transport algorithm is to combine two numerical schemes, a low order scheme (of which Godunov's scheme is a classic example) which has no numerical overshoots or oscillations but a smeared shock, and a high order scheme with a high shock resolution but with numerical oscillations and overshoots in the solution. The combination is performed in such a way that where numerical oscillations occur in the high order solution the combination is biased towards the low order solution but elsewhere the combination is biased towards the high order solution. Essentially the diffused low order solution is only used to the extent that is necessary to stabilise the solution. The logic by which it predicts a numerical oscillation is by presuming that any oscillation in the high order solution that is not in the low order solution or in the solution at the previous timestep (U^n) is numerical. The flux-corrected transport algorithm is able to capture a shock within two elements without numerical overshoots.

The high order scheme is simply the Taylor-Galerkin scheme described above and the low order scheme is essentially the same scheme only using a lumped mass matrix when solving the matrix

equations and with diffusion added. The low order solution therefore does not accurately resolve discontinuities but this is compensated by the high order scheme.

2.7.1 The low order scheme

The low order Taylor-Galerkin scheme used by Löhner[28] is expressed as follows

$$M_L \Delta U^l = RHS + DIFF \tag{28}$$

where U^l is the low order solution and

$$DIFF = c_d(M_C - M_L)U^n \tag{29}$$

where c_d is the diffusion coefficient. The low order solution can then be calculated as follows

$$\Delta U_j^l = (M_L)_j^{-1}(\sum_e (RHS + c_d(M_c - M_L)_e U_j^n)_j \tag{30}$$

In Peraire et al[20] a slightly more sophisticated diffusion model is used

$$DIFF = c_d S_e(M_C - M_L)U^h \tag{31}$$

where S_e is the average value of the nodal values S_i (calculated below) for each element (S_e and S_i being unrelated to S, the source term used earlier).

$$S_i = \frac{|\sum_e (M_c - M_L)p_i|_i}{(\sum_e |(M_c - M_L)_e p_i|)_i} \tag{33}$$

S_i represents the degree to which the nodal value represents a local extremum and varies between 0 and 1. The main difference between these two diffusion models is that the factor S_i has the effect of concentrating the diffusion at extrema in the solution and limiting it elsewhere. It can be regarded as a diffusion limiter. However as this model is used in conjunction with a FCT algorithm which in

effect remove excess diffusion this factor would presumably be redundant. Both these models were tested with the FCT algorithm and no difference in solution was observed. As the first diffusion model is computationally quicker it was used for all the tests done in this work. However if a FCT algorithm was not being used and diffusion was used on its own to stabilise the shock without being used in conjunction with a high resolution scheme, the second model would give better shock resolution..

In this work, a diffusion coefficient value of $c_d = 1.5$ was used. The value used is not critical as long as it is sufficient to suppress numerical overshoots and not excessive as to cause the solution to break down. As long as the coefficient is between these extremes, variations will not be observable in the final solution as the flux-corrected transport algorithm effectively eliminates excess diffusion in the low order solution by biasing the solution towards the high order solution.

2.7.2 The FCT algorithm

In the following algorithm, reference is made to the element contributions which are the unassembled element solutions from the high and low order schemes. The flux-corrected transport algorithm consists of the following steps :

1. Compute the high order element contributions (HEC) using the high order scheme ($\mathbf{U}^h$).

2. Compute low order element contributions (LEC) from the low order scheme ($\mathbf{U}^l$).

3. Compute the antidiffusive element contributions (AEC = HEC - LEC).

4. Limit the AEC^c :

$$AEC^c = C_e AEC$$

5. Add the limited anti-diffusive element contributions to the low order solution obtaining the final solution :

$$U^{n+1} = U^{l} + \sum_{e} AEC^{e}$$

The crucial part of the algorithm is the calculation of the limiter C_e. The limiter is based on one or a combination of the solution variables. Here two limiters were calculated based on ρ and ρe. The final limiter used is the minimum of the two limiters.

$$C_e = \min\ (C_\rho\ ,\ C_{\rho e})$$

First a value is calculated representing the maximum and minimum values the chosen solution variable is allowed to reach at a node for no numerical overshoots to appear in the solution. For any node this value is determined by the maximum value of both the solution at time n and the low order solution at time n+1 of all the nodes surrounding that node.

$$U_i^{max} = \max(U_i^{l},\ U_i^{n})$$
$$U_i^{min} = \min(U_i^{l},\ U_i^{n})$$

The maximum and minimum nodal value of each element is then calculated

$$U_e^{max} = \max(U_A,\ U_B,\ ....,\ U_N)$$
$$U_e^{min} = \min(U_A,\ U_B,\ ....,\ U_N)$$

where A, B , ..., N are the nodes of element e.

$$U_i^{max} = \max(U_1^{max},\ U_2^{max},\ ....,\ U_n^{max})$$
$$U_i^{min} = \min(U_1^{min},\ U_2^{min},\ ....,\ U_n^{min})$$

where 1, 2, ..., n represents the elements surrounding node i.

The value U_i^{max} and U_i^{min} are the maximum and minimum of the low order and previous solution of all the nodes surrounding node i. They represent the maximum and minimum values respectively that the solution is allowed at node i to achieve without introducing overshoots into the solution. The

limiter which limits the anti-diffusive element contributions so that these values are not exceeded is calculated below.

The following parameters need to be defined :

P_i is the sum of all positive/negative antidiffusive element contributions to node i :

$$P_i^+ = \sum_e \max(0, AEC_e)$$
$$P_i^- = \sum_e \min(0, AEC_e)$$

Q_i is the maximum increment/decrement that can be added to the low order solution without creating overshoots/undershoots.

$$Q_i^+ = U_i^{max} - U^l$$
$$Q_i^- = U_i^{min} - U^l$$

A ratio R is obtained which is the ratio of allowable AEC to the originally calculated AEC (step 3).

if $P^+ > 0$ and $P^- < 0$

$$R^+ = \min(1, Q^+/P^+)$$
$$R^- = \min(1, Q^-/P^-)$$

else if $P^+ = 0$ and $P^- = 0$

$$R^+ = 0$$
$$R^- = 0$$

where R is the ratio of the allowable AEC to the actual AEC.

Finally

$$C_e = \min(element\ nodes)\begin{cases} R^+ , & if\ AEC > 0 \\ R^- , & if\ AEC < 0 \end{cases}$$

The limiter C_e limits the antidiffusive element contributions such that no overshoots are obtained. From experience in this work, it has been found that while this successfully stabilises the solution, still some non physical numerical oscillations are found in the solution. These are very small (typically in the region of 1 - 2% of the solution) however contour plots are very sensitive to such oscillations and these may commonly be seen in diffraction patterns. The following modification is therefore proposed which biases the solution slightly more to the low order solution

$$C_m = kC_e$$

where k is typically set to 0.95 - 0.98. C_m is then used in place of C_e in step 4 of the flux-corrected transport algorithm. This has been found to minimise numerical oscillations without affecting resolution too adversely.

2.8 Steady and Unsteady Solutions

A steady state solution is obtained by first setting the initial values of the solution to freestream conditions and then using the transient algorithm to converge the solution to steady state. The solution is considered to be converged when the maximum density change at all nodes over one timestep does not exceeds a certain tolerance. For most of the problems solved here, this tolerance was set to 0.5%. This was found to be a compromise between accuracy and excessive computational time to converge to this tolerance. As an accurate transient solution is not required when converging to steady state, β in the stability criterion (26) is set to 0.9 as opposed to 0.4 and only one iteration of the matrix solver is performed as opposed to three iterations.

Unsteady solutions, generally involving a normal shock impinging on some geometrical feature is obtained by dividing the domain into two regions, a pre shock and a post shock region. The initial conditions before and after the shock are then defined within each region.

3. AUTOMATIC MESH GENERATION

3.1 Introduction

A fully automatic mesh generator was implemented for the generation of two dimensional triangular meshes. The primary purpose of automatic mesh generation is to reduce the amount of data input that the analyst has to perform. All the analyst has to do when using an automatic mesh generator is to describe the boundaries of the domain to be meshed, specify the desired mesh resolution and the mesh generator will then proceed to subdivide the geometry into triangles of the specified size. Automatic mesh generation is more than just a luxury. If optimising algorithms in an attempt to reduce computer time is a subject of intense research, reducing operator time, which nowadays is considerably more expensive, should be considered as at least as important.

Two of the most widely used approaches for automatically generating triangular meshes are Delauney triangulation[31] and the advancing front method[32-34]. The Delauney approach triangulates an existing set of nodal points in a consistent and unique manner, that is, for a given set of nodal points, there is only one valid Delauney triangulation. The advancing front method, a more heuristic approach to mesh generation, was originally developed to triangulate an existing set of coordinates[32] but later it was modified to create nodes concurrently with the generation of elements[33]. Either method could have been used, but the advancing front method was chosen due to its ability to generate nodes and elements concurrently, avoiding the need for a separate algorithm for the generation of nodal points.

3.2 The Advancing Front method

In this work the advancing front algorithm of Peraire et al[33] was used. In this paper, the algorithm was used for adaptive remeshing but in this work it was only used to create the initial mesh. The procedure starts with the creation of a generation front. The front is a set of line segments initially forming the boundary of the domain and later, once the generation of elements has begun, forms the boundary between the elements already generated and the region still to be meshed. The front is hence updated after the creation of each new element. Segments from the front form the base of new elements to be generated. The front advances into the domain until the region is completely meshed and the front vanishes.

This routine is used to create a uniform mesh with the average element side length = δ. However this is not a fundamental restriction of the algorithm. The algorithm can be used to generate meshes where δ varies throughout the mesh, which is appropriate when adaptively regenerating the mesh.

The procedure proceeds basically as follows :

(1) The boundary curves are divided into a set of line segments each equal in length to δ.

(2) A segment from the generation front is selected to form a base for a new element.

(3) A list of nodes is created representing possible connectivities that can be used to form a triangular element with that base. This list is ordered preferentially such that the node that represents the best possible connectivity is first in the list.

(4) A validity check is performed on each of the potential elements that can be created using the nodal list. The first node in the list that forms a valid element is used to create a new element.

(5) The front is updated. If at this point the front is empty, the process is complete, otherwise the procedure goes back to (2)

(6) The completed mesh is smoothed in order to improve the shape of the elements.

3.2.1 Creation of the initial generation front

The boundary of the domain is described by the user by a set of linear, quadratic and cubic parametric curves. The traditional finite element parametrisation[37] of the curves are used, so that all the user has to do to define a cubic curve for example is do give the four coordinates which the curve interpolates. A boundary code (solid wall, farfield) is supplied by the user for each curve which the mesh generator later applies to the elements generated along that curve.

Each curve is divided into directed line segments, each segment having length δ. The way each line segment is directed is important. The algorithm uses each front segment as a base of a new element. The algorithm needs some way of determining on which side of the line segment an element needs

to be created, remembering that what is obvious to the human observer is in no way obvious to the computer. The computer determines this by always generating an element to the left of the directed line segment when facing in the direction of the line segment. This can be seen from fig 3.1. The line segments are therefore directed so that on external boundaries they go anticlockwise around the geometry and on internal boundaries, that is holes in the domain, clockwise. If this were incorrectly done, the mesh generator could start generating elements on the exterior of the domain, or on the interior of what are supposed to be holes in the domain. In all the figures in this chapter, the direction of each front segment are indicated with arrows.

3.2.2 Selection of a front segment

One of the front segments must be selected to form the base of a new element. When elements of different sizes are being created, the smallest segment is usually selected (this prevents large elements crossing the path of smaller elements). As only uniform meshes are of interest here any segment can be chosen. It is convenient, purely from the point of view of data management, to select the last segment recorded in the front.

3.2.3 Creation of a nodal list

In this step a list of nodes are created each of which represent possible connections in order to generate a triangular element using the previously selected front segment as a base. The nodes are listed in order of preference, so that the first node in the list is the preferred node.

First an ideal node C is created (fig 3.1). This is a node such that the length of each of the new element sides that could potentially be formed with this node is equal to δ. Then a list of nodes from the front within some radius r (r is typically = δ) of C is created and ordered such that the first node is closest to C. The node C is placed at the head of the list unless

$$AN_1 < 1.5\,\delta \text{ and } BN_1 < 1.5\,\delta$$

where A and B are the nodes of the front and N_1 is the first node in the list, in which case C will be placed second in the list. This means that the ideal element is the preferred element unless there is an existing node that can be selected that will form a well shaped element (prescribed by the above condition). As one proceeds down the list, the elements created will be progressively distorted from the ideal.

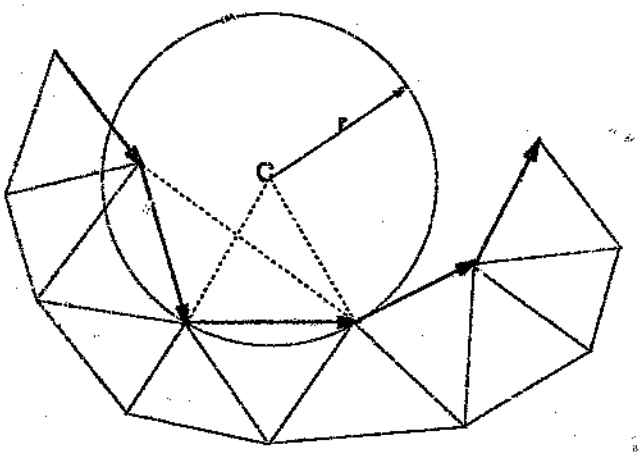


Figure 3.1 Potential new elements (dashed lines)

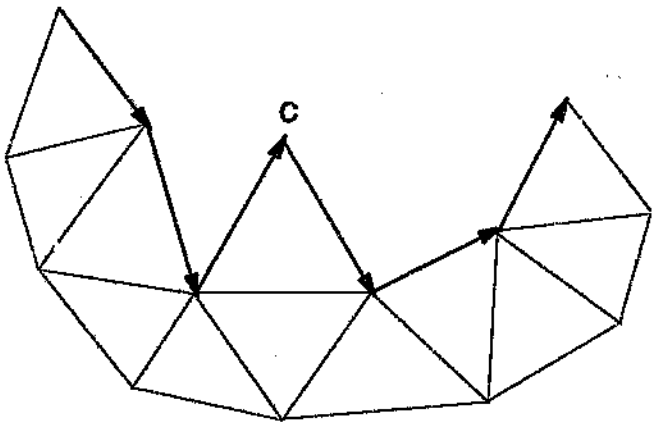


Figure 3.2 Updated front - ideal node selected

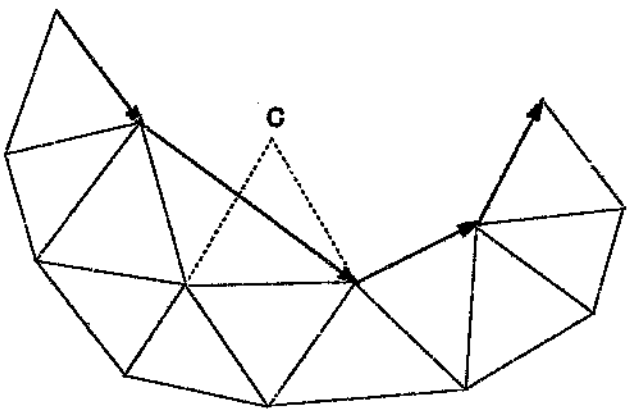


Figure 3.3 Updated front - existing node selected

3.2.4 Validity check

Before an element can be created using one of the nodes in the above list it has to pass a validity check. This is to ensure that any element created does not intersect any other existing element in the mesh (though element sides are naturally allowed to coincide, see fig 3.3). The validity check is performed by checking whether any of the two new sides created do not intersect any side of the front. (Fig 3.4) This is adequate because none of the sides could intersect any other element without passing through the front, the front being the boundary between the unmeshed region and generated elements. Only intersections are forbidden. Nodes and element sides are allowed to coincide. The first node which passes the test is used to create the new element.

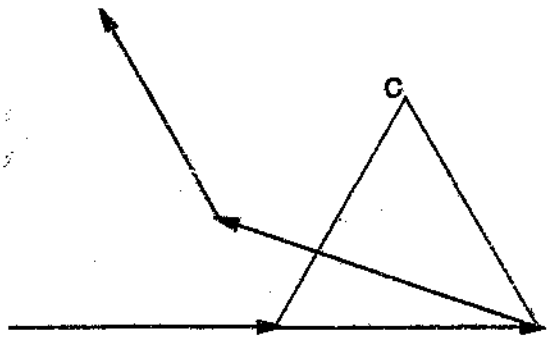


Figure 3.4 Invalid element intersecting front

3.2.5 Updating the front

Once a new element has been accepted and created the front has to be updated to reflect the inclusion of the new element. First the front segment forming the base of the new element must be deleted, as it is now no longer on the boundary between the meshed and unmeshed domain. Next it must be determined whether any of the new element sides created coincide with any of the front segments. If it does the appropriate front segment must be deleted from the front. The updated fronts are shown in figs 3.2 - 3.3. If at this stage the front is empty the mesh generation is complete. If not the algorithm proceeds to step 3.2.2.

3.2.6 Mesh smoothing

Though not an essential process of the mesh generation, mesh smoothing does enhance the quality of the mesh. The mesh generated in the previous steps will quite likely have elements which may be quite distorted from the ideal. Mesh smoothing proceeds by moving each node, with the exception of boundary nodes, to the centre of its surrounding nodes. Two passes of mesh smoothing are performed after which the process converges and further smoothing has little effect. (see fig 3.5f)

3.3 Automatic mesh regeneration

Though mesh regeneration is not used in this work, the above algorithms extension to it will be described.

In the above description, a single value of δ is used throughout the generation of the mesh. A different value of δ could be used however for every element generated. Typically when adaptively regenerating the mesh δ would be determined from an error analysis performed on a previous mesh. The value of δ would then be calculated to attempt to create a uniform error distribution throughout the mesh. The error estimate is generally taken to be a function of the second derivative of density. This results in the following requirement for the mesh size h (shown for single dimension).

$$h^2 \left| \frac{d^2\rho}{dx^2} \right| = C$$

where C is the desired constant error.

Automatic mesh regeneration is typically applied to steady state problems where a new mesh may only be created two or three times. This is due to the fact that mesh regeneration is a computationally expensive process. In Probert et al[34] partial adaptive remeshing is used where instead of the entire mesh being regenerated only those areas where the error goes out of predefined bounds is the mesh regenerated.

One problem with adaptive remeshing is that the projection of the old mesh values onto the new mesh is not exact. This is not a serious problem in steady state problems where an accurate transient

solution is not required but it does result in a loss of conservation in transient problems, affecting for example the speed of a shocks in the solution. Until this problem is effectively addressed, it would be difficult to justify adaptive remeshing for transient problems.

3.4 Three Dimensional Mesh Generation

The above technique is readily extended to three dimensional tetrahedral meshes and is done so in Peraire et al[20]. In the three dimensional case, the generation front is a set of planar triangles each of which can form the base of a tetrahedral element. The initial triangular front is typically created by a two dimensional triangular mesh generator adapted to mesh surfaces. In the validity check, each face of the potential tetrahedron has to be checked for intersection against any of the triangular boundary front faces.

In this work, a three dimensional automatic mesh generator has not been implemented. Traditional parametric mapping techniques[29-30] are used to generate the three dimensional meshes. This method is described in appendix B.

3.5 Automatic Mesh Generation Example

In figures 3.5 (a)-(f) various stages of the generation of a mesh for a rectangular plate with two circular holes in it are shown. Though a fairly simple geometry it would still be fairly time consuming to divide this geometry into quadrilateral blocks for parametric mesh generators. The behaviour of the advancing front as it progresses into the domain until it vanishes is clearly demonstrated.

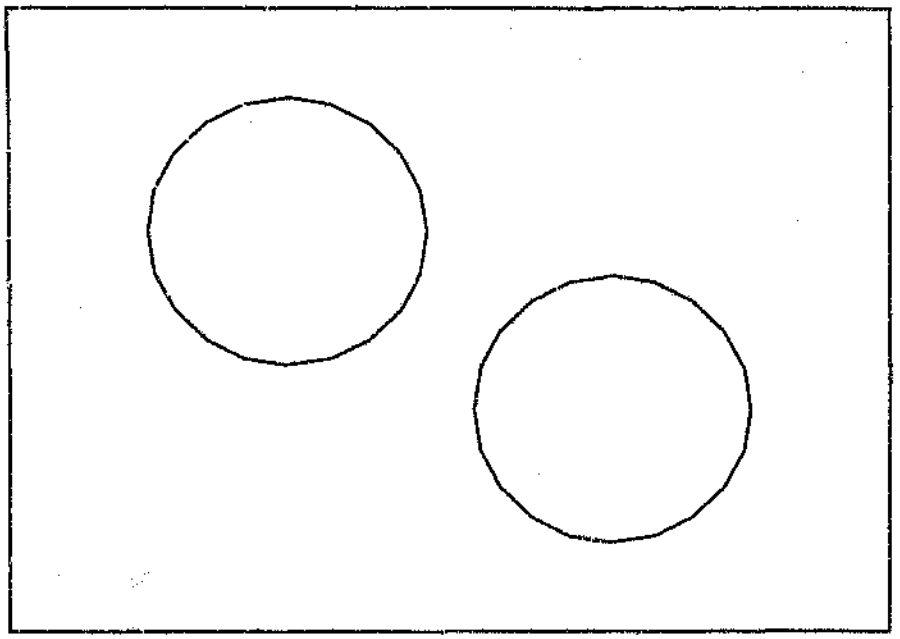


Figure 3.5 (a) Initial generation front forming boundary of geometry.
Number of fronts = 106

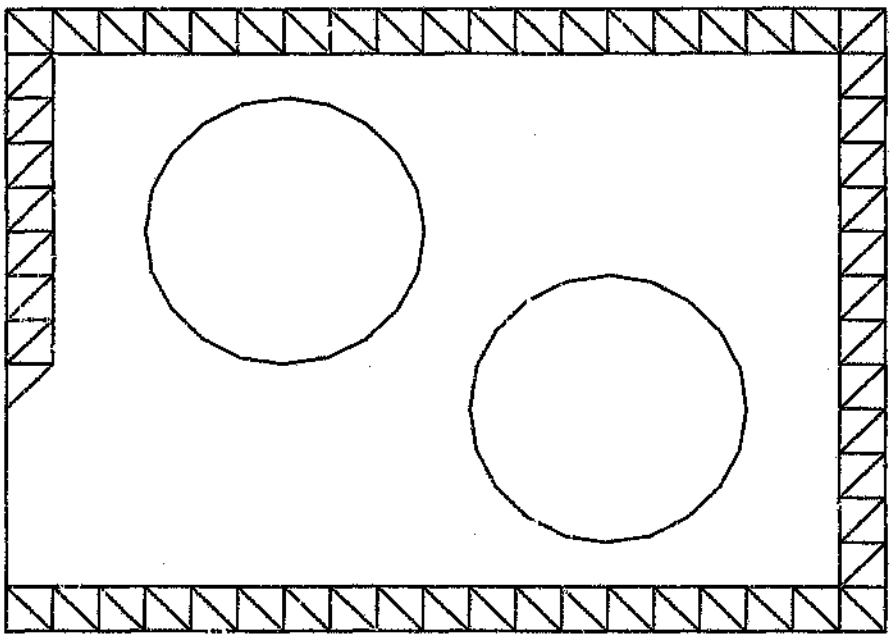


Figure 3.5 (b) Number of fronts = 99

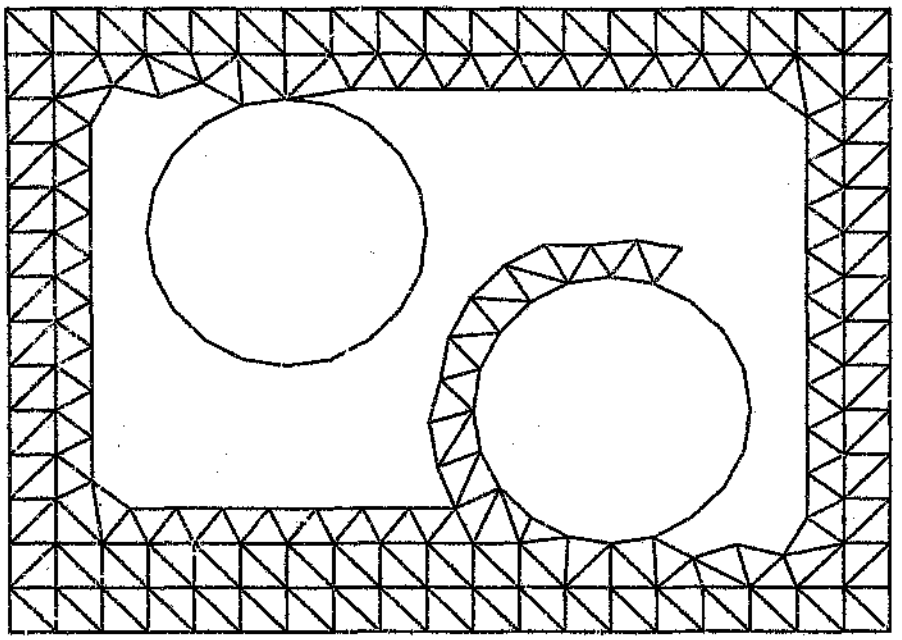


Figure 3.5 (c) Number of fronts = 82

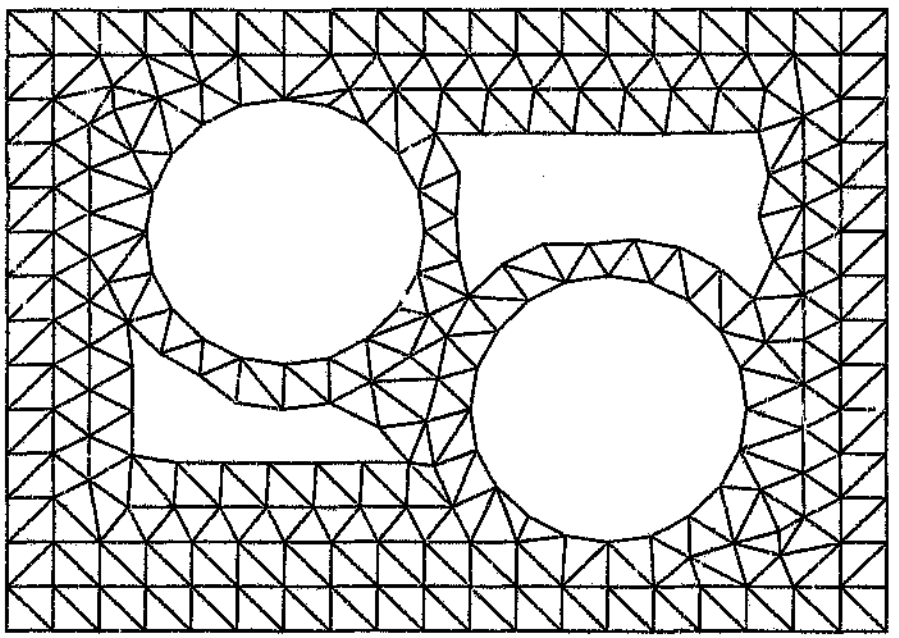


Figure 3.5 (d) Number of fronts = 38

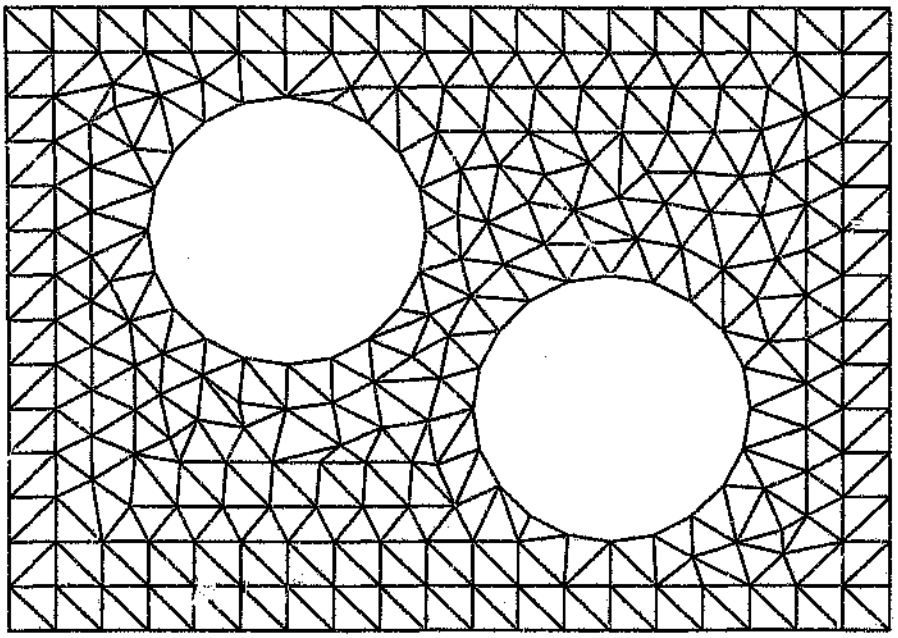


Figure 3.5 (e) Number of fronts = 0, Number of elements = 466

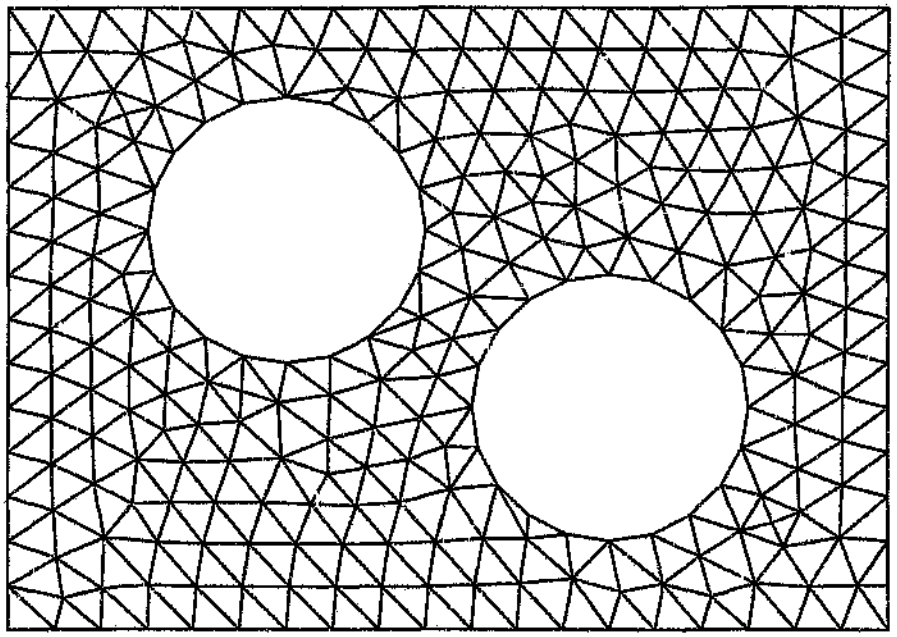


Figure 3.5 (f) Completed mesh after two passes of smoothing

Figure 3.5 Demonstration of automatic mesh generation

4. ADAPTIVE REFINEMENT

4.1 Introduction

One characteristic of compressible flows is that within the flow domain very high gradients tend to exist in conjunction with relatively shallow flow gradients, for example, a shock wave may exist in an otherwise smooth flowfield. For the areas with high flow gradients, a fine mesh is required to be able to accurately model those features while for the shallow gradients a relatively coarse mesh will suffice. It is not practical to manually optimise the mesh as the solution changes continuously with time, so the usual approach is to create a uniform fine mesh over the whole flow domain. Where uniformly fine grids are used, the computational expense often becomes excessive so a lower overall resolution has to be used, resulting in a poorer solution quality. This problem is dealt with by using automatic adaptive mesh techniques where the computer repeatedly adapts the mesh in an attempt to maintain optimal mesh resolution.

An adaptive refinement technique works essentially as follows. An initial mesh is created using the automatic mesh generation techniques described in chapter 3. An error analysis is performed on the initial solution on the mesh and the mesh is refined accordingly. Every few timesteps, typically 5 to 10, the error analysis is repeated and the mesh again adapted. The approach implemented here is the adaptive refinement algorithm developed by Löhner[35]. Adaptive refinement techniques refine the mesh by subdividing existing elements into smaller elements where appropriate and coarsening it again by recovering the original element when a high resolution is no longer needed, that is when the high gradient feature has passed. This is as opposed to adaptive remeshing techniques where a mesh is regenerated using an automatic mesh generator.

Parts of the algorithm description is performed with the aid of pseudocode. This is not meant to resemble any computer language in particular but is just used to help illustrate the algorithm.

There are three main steps to this algorithm :

- Perform an error analysis on the solution on an existing mesh.
- Refine the mesh where the error indicator exceeds some specified value.
- Coarsen the mesh where the error drops below a specified value.

4.2 Error Indicator

In order for the mesh to be able to appropriately refine and coarsen itself, an estimate of the solution error on the current mesh is necessary. The error indicator described by Löhner[35] is used here. It is common for these types of problems for the error indicator to be a function of the second derivative of some flow variable. Density is generally used, as an error indicator based on pressure will not pick up contact discontinuities and sliplines. Predefined bounds are prescribed by the user for the error indicator. If the error exceeds the upper bounds, the mesh is refined. If the error drops below the lower bounds, the mesh is coarsened.

The error indicator is first developed here in one dimensional finite difference form in order to demonstrate the concepts involved. The second derivative can be used as the basis of an indicator of the error as follows :

$$E_i = h^{-2}|U_{i+1} - 2U_i + U_{i-1}|$$

It is desirable for the error indicator to be dimensionless and also to be able to use preset error bounds for a wide class of problems. The above indicator will not fill this criterion. Where strong and weak shocks exist in the same solution it is possible that only the strong shocks will be picked up. The appropriate error bounds will therefore vary from problem to problem and even at different areas of the same mesh. It is for this reason that a normalised error indicator based on a H2 seminorm is used, which is obtained by dividing the second derivative by the absolute maximum value of the second derivative. This will eliminate the effect of magnitude of the feature on the error indicator, resulting in strong and weak shocks returning similar error values.

$$E_i = \frac{|U_{i+1} - 2U_i + U_{i-1}|}{|U_{i+1} - U_i| + |U_i - U_{i-1}|}$$

Eliminating the actual magnitude from the indicator does have the problem that very small numerical oscillations regardless of magnitude could return a large error. It is for this reason that a noise filter is included in the error indicator in order to eliminate the effect of very small features on the error indicator. The modified form of the error indicator with the noise filter is expressed below.

$$E_i = \frac{|U_{i+1} - 2U_i + U_{i-1}|}{|U_{i+1} - U_i| + |U_i - U_{i-1}| + \epsilon(|U_{i+1}| + 2|U_i| + |U_{i-1}|)}$$

The terms following ε are the noise filter. The appropriate value of ε can vary. Initially for perfect gas flows a value of 0.05 was used and found to work well for stronger shocks but weak reflected shocks were found to be filtered out as noise. A value of 0.005 was found to work better over stronger and weaker shocks. For liquid shocks, it was found that using this value resulted in shock waves been dismissed as noise by the error indicator. The reason for this was that liquid shock waves cause a very small change in density of the liquid (< 1% is common) . A value for ε of 0.0005 was found to be more suitable for liquid shock problems.

The above error indicator expressed in multidimensional finite element notation is as follows :

$$E_i = \sqrt{\frac{\sum_{k,l} \left(\int_\Omega \frac{\partial N_i}{\partial x_k}\frac{\partial N_j}{\partial x_l} d\Omega\, U_j\right)^2}{\sum_{k,l} \left(\int_\Omega \left|\frac{\partial N_i}{\partial x_k}\right| \left[\left|\frac{\partial N_j}{\partial x_l} U_j\right| + \epsilon\left(\left|\frac{\partial N_j}{\partial x_l}\right| |U_j|\right)\right] d\Omega\right)^2}}$$

where k and l represent the different coordinate directions and subscripts i and j represent nodal values. The above error indicator will return a value between 0 and 1. In all the problems solved in this work the upper error bounds was set to 0.3 and the lower error bounds was set to 0.1. The fact that the same error bounds could be used for a wide variety of problems indicate the advantage of the above form of the error indicator.

4.3. Mesh management

For steady flow problems it is not necessary to coarsen the mesh at any stage and any refinement algorithm developed exclusively for steady flows would therefore be fairly simple to implement. However in transient problems, as flow features change position it is desirable to be able to coarsen the mesh again. Therefore enough information must be stored to be able to recover the initial mesh.

One of the basic concepts of adaptive refinement algorithms is that of refinement levels. All elements on the initial grid are at level 0 which is the coarsest level allowable. If any of these elements are refined, that is, subdivided into four, these four elements are then at level 1. If these elements are again refined, the subsequent elements will be at level two et cetera.

This adaptive refinement algorithm maintains mesh consistency at all times. Examples of consistent and inconsistent meshes are shown in fig 4.1. Where the mesh is usually refined by dividing elements into 4 smaller elements, consistency is maintained by transitional elements which are the result of an element being divided into 2 elements. It is not essential to maintain a consistent mesh however. If an inconsistent mesh is used, it is possible to keep the solution consistent through the use of constraint equations in the solver.

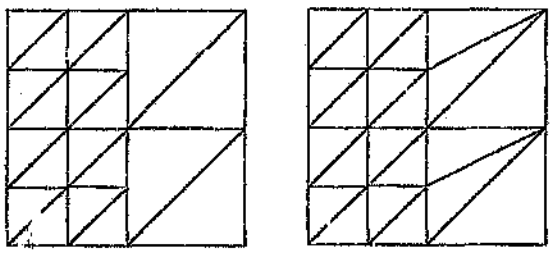


Figure 4.1 Inconsistent mesh (left) Consistent mesh (right)

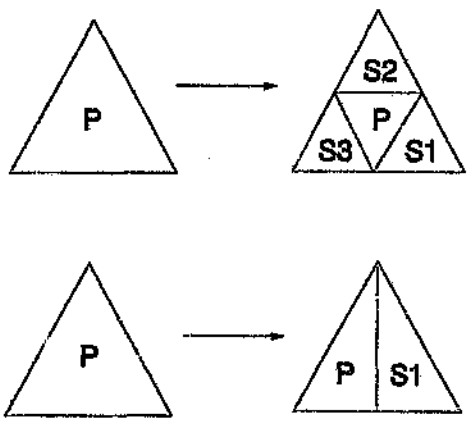


Figure 4.2 Parent and son elements

When an element is subdivided into four elements, the central element gets the element number of the original element and is referred to as the parent element (fig 4.2). The surrounding three elements are referred to as the sons of that parent element and are added to the end of the element array. These sons can later become parent elements themselves. When an element is divided into two, that is, transition elements are created, one element, any of the two, is labelled the parent element and the other the son. Only one pass of refinement or coarsening is performed each time the algorithm is called.

Some of the variables and data structures used in this algorithm will now be described :

NELMS	No of elements in mesh
NNODES	No of nodes in mesh
ELEMENTS (1..NELMS, 1..3)	The three node numbers for each element
COORDS(1..NNODES, 1..2)	The x and y coordinates of each node
ERROR(1..NNODES)	The error indicator calculated for each node in sec 4.2

MESHTRACK(1.. NELMS, 1..8)

This array is created which keeps track of the mesh history and contains adequate information in order to retrace the refinement of the mesh. For each element in the mesh the following information is kept in this array.

(1 - 3) This is the list of sons from a refinement. If an element has no sons these will all be set to zero. Only sons that have not since become parents themselves are stored. If a parent has only one son as a result of a 1:2 refinement the son is stored in (1) and (2 - 3) are set to zero. If (1 - 3) are all non zero the algorithm knows that these son elements have not been refined further and are available to be coarsened if necessary.

(4) This is the number of the parent element from which the current element originated.

(5) This is the orientation relative to the parent element from which this element was created. The value will be 1,2 or 3 depending on its position, see fig 4.2.

(6) The refinement level of this element.

(7) Refinement code. This is to distinguish between elements involved in normal 1:4 refinements and transitional elements created from 1:2 refinements. This is needed as a different logic is used in these different cases. 0 = conventional, non transitional element, 1 = parent of a 1:2 refinement, 2 = son of a 1:2 refinement.

(8) Refinement level at which this element was originally created.

4.4 Mesh refinement

The first step in refining the mesh, is to determine which elements need to be refined. This is done by marking all elements that have a node whose error value exceeds that specified by the above error indicator.

REFELM(1..NELMS) = 1 if element is to be refined, else = 0

In transient problems it is also necessary for the mesh to be refined around areas of high flow gradient so that as a feature (typically a shock wave) moves, it does not move outside the refined area into an area with coarser mesh resolution. This enables refinement to be carried out every few timesteps (5 -10) as opposed to every timestep. For steady state problems no protective layers are necessary as refinement is only carried out once the solution has converged to steady state. If the converged solution is valid the position of flow features should not change once refinement has occurred. However one layer of protection is generally allowed for steady state problems. The following procedure is followed for the addition of protective layers.

- Loop over all elements marked for refinement and mark all the nodes connected to those elements.

- Loop over all elements. If any node in those elements was marked in the previous step that element must be marked to be refined, unless it is already at the maximum specified refinement level.

(6) The refinement level of this element.

(7) Refinement code. This is to distinguish between elements involved in normal 1:4 refinements and transitional elements created from 1:2 refinements. This is needed as a different logic is used in these different cases. 0 = conventional, non transitional element, 1 = parent of a 1:2 refinement. 2 = son of a 1:2 refinement.

(8) Refinement level at which this element was originally created.

4.4 Mesh refinement

The first step in refining the mesh, is to determine which elements need to be refined. This is done by marking all elements that have a node whose error value exceeds that specified by the above error indicator.

REFELM(1..NELMS) = 1 if element is to be refined, else = 0

In transient problems it is also necessary for the mesh to be refined around areas of high flow gradient so that as a feature (typically a shock wave) moves, it does not move outside the refined area into an area with coarser mesh resolution. This enables refinement to be carried out every few timesteps (5 -10) as opposed to every timestep. For steady state problems no protective layers are necessary as refinement is only carried out once the solution has converged to steady state. If the converged solution is valid the position of flow features should not change once refinement has occurred. However one layer of protection is generally allowed for steady state problems. The following procedure is followed for the addition of protective layers.

- Loop over all elements marked for refinement and mark all the nodes connected to those elements.

- Loop over all elements. If any node in those elements was marked in the previous step that element must be marked to be refined, unless it is already at the maximum specified refinement level.

The above two steps are repeated for every protective layer to be added The above algorithm is now illustrated using pseudocode.

```
MARKER(1..NNODES)      Integer array. Initially set to contain zeros.

FOR I = 1 TO NELMS
 IF REFELM(I) = 1 THEN                    {Mark all nodes connected to elements to be
                                           refined}
    MARKER(ELEMENTS(I, 1)) = 1
    MARKER(ELEMENTS(I, 2)) = 1
    MARKER(ELEMENTS(I, 3)) = 1
 ENDIF
NEXT I

FOR I = 1 TO NELMS                        {Mark all elements connected to nodes in the above
 IF MARKER(ELEMENTS(I, 1)) = 1 OR          loop for refinement}
    MARKER(ELEMENTS(I, 2))  = 1 OR
    MARKER(ELEMENTS(I, 3)) = 1 THEN
    REFELM(I) = 1
 ENDIF
NEXT I
```

All the elements that need to be refined have now been marked. However it is not possible to just go ahead and refine these elements as the mesh will then no longer be consistent. Only a certain number of refinement cases are allowed. Any of the elements to be refined must be refined in one of the ways shown in figure 4.3.

Only a limited number of refinement cases are allowed. These are shown in figure 4.3. More data structures need to be developed at this point. First an array of all the element edges in the mesh is needed, which gives for each edge, the two nodes that it connects. Secondly, an array needs to be created which for each element gives the three edges that are connected to it.

```
EDGES(1..NEDGES, 1..2)        The two node numbers defining each edge
```

ELMEDGES(1..NELMS, 1..3) The three edges for each element, this references the EDGES array..

Two adjacent elements will therefore reference the same edge.

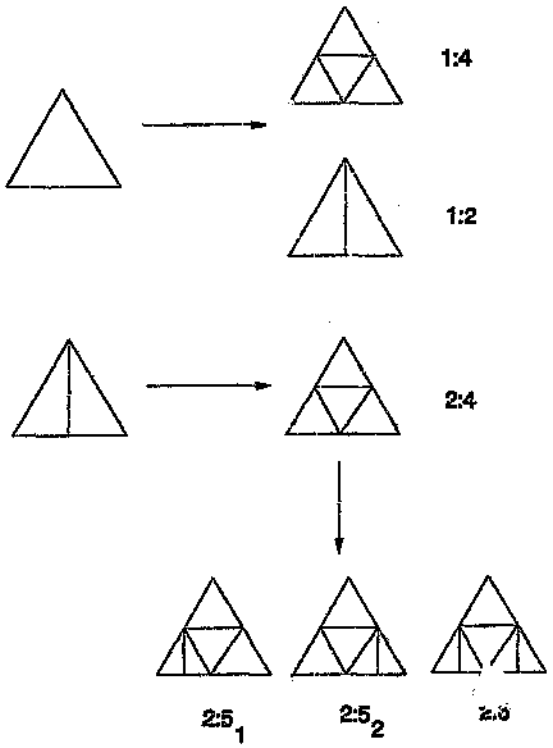


Figure 4.3 Allowable refinement cases

The algorithm then proceeds as follows.

A loop is performed around all the elements. All edges belonging to elements that have previcusly been marked for refinement, are marked as needing to be refined (using array REFEDGE). By knowing which edges need to be refined, it now becomes possible to determine which other elements need to be refined in order for mesh consistency to be maintained.

```
FOR I = 1 TO NELMS                    {This loop marks edges for subdivision}
  IF REFELM(I) = 1 THEN
     REFEDGE(ELMEDGES(I, 1)) = 1
     REFEDGE(ELMEDGES(I, 2)) = 1
```

```
    REFEDGE(ELMEDGES(I, 3)) = 1
  ENDIF
NEXT I
```

Another loop is now performed around all the elements. For each element the number of sides that need to be refined are checked. If no sides are to be refined, then that element does not need to be refined. If three edges are to be refined, a 1:4 refinement is possible. If only one edge is to be refined, a 1:2 refinement is possible. The problem comes in when two edges need to be refined, as an appropriate refinement (1:3) is not possible. The way that this is dealt with is to mark the edge that is not to be refined for refinement converting it to a 1:4 refinement. Of course by marking an extra edge for refinement, this has a cross effect on adjacent elements. If any new edges are specified for refinement the above loop has to be performed again to check whether valid refinements are still possible.

```
FOR I = 1 TO NELMS
  COUNT = 0
  FOR J = 1 TO 3
    IF REFEDGE(ELMEDGES(I, J)) = 1 THEN
      COUNT = COUNT + 1
    ENDIF
  NEXT J
  IF COUNT = 2 THEN                    {Convert to 1:4 refinement}
    REFEDGE(ELMEDGES(I,1)) = 1
    REFEDGE(ELMEDGES(I,2)) = 1
    REFEDGE(ELMEDGES(I,3)) = 1
  ENDIF
NEXT I
```

A new node is created on the midpoint of any edge that has been marked for refinement. The existing solution is interpolated between the two points defining the edge.

Similar logic is used for 1:2 refinements. If any transitional element is to be further refined, it first undergoes a 2:4 refinement.

A loop around the elements is performed. For each element that has refined sides, the appropriate refinement subroutine is called (1:2 , 1:4 , 2:5_1 , 2:5_2 , 2:6).

4.5 Mesh Coarsening

For an element to be coarsened, the following criteria must be fulfilled.

- The parent and its three sons must be at the same refinement level. This in effect means that neither the parent or son elements may have sons at a higher refinement level. This will limit the coarsening to either 4:1 or 2:1 coarsening cases (fig 4.4)

- The error indicator of all the nodes belonging to the parent and sons must be below the required coarsening tolerance.

- An inconsistent mesh must not be created by the mesh coarsening.

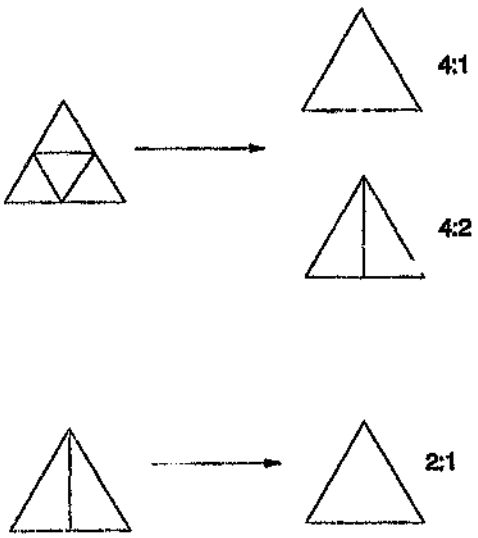


Figure 4.4 Allowable coarsening cases

An element that has previously been divided into four may be coarsened again to recover the original element. Firstly only one level of coarsening may be performed at a time. This means that no sons of the parent element that is to be coarsened may have sons themselves. If they do, they would have be coarsened at the higher refinement level first. The parent and son elements must therefore be at

the same level of refinement. All nodes that belong to the parent and sons must be specified by the error indicator as requiring coarsening.

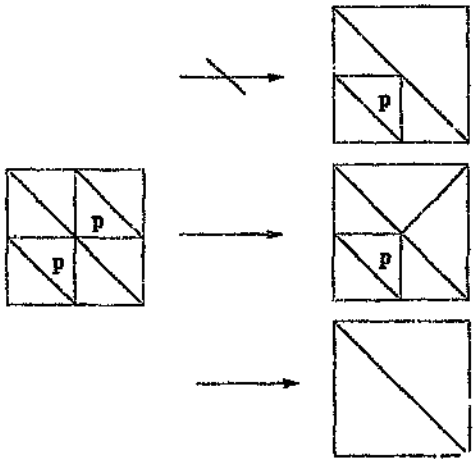


Figure 4.5 Maintaining consistency when coarsening

For a 4:1 coarsening to occur, all nodes belonging to the parent element must be deleted. However for the mesh to remain consistent, the adjacent parent element must also be coarsened as the two parents share a common node (see fig 4.5). This is ensured as follows. All nodes belonging to a parent element to be coarsened is marked. If a node is marked twice, this means that both adjacent parent elements connected to that node are to be coarsened. Only those nodes that are marked twice may be deleted. If a node is only marked once then that parent element is not allowed to be coarsened. For a parent element to be coarsened either all three of its nodes or two of its nodes must be deleted (allowing the 4:2 transition coarsening case). If a case is found that only one node is to be deleted, coarsening is not allowed.

This can be demonstrated algorithmically as follows :

DELELM(1..NELMS) = 1 if element is a parent to be deleted.

```
FOR I = 1 TO NELMS
  IF DELELM(I) = 1 THEN          {Mark all nodes connected to parent elements to be deleted}
     MARKER(ELEMENTS(I, 1)) = MARKER(ELEMENTS(I, 1)) + 1
     MARKER(ELEMENTS(I, 2)) = MARKER(ELEMENTS(I, 2)) + 1
```

the same level of refinement. All nodes that belong to the parent and sons must be specified by the error indicator as requiring coarsening.

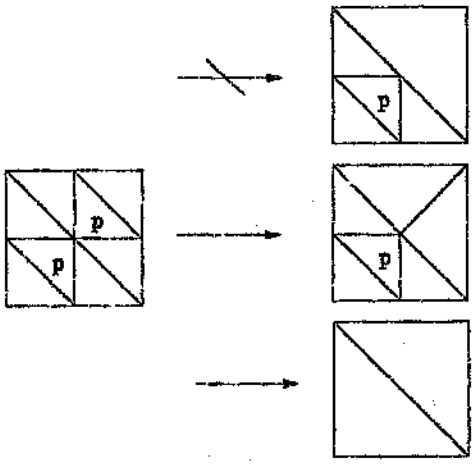


Figure 4.5 Maintaining consistency when coarsening

For a 4:1 coarsening to occur, all nodes belonging to the parent element must be deleted. However for the mesh to remain consistent, the adjacent parent element must also be coarsened as the two parents share a common node (see fig 4.5). This is ensured as follows. All nodes belonging to a parent element to be coarsened is marked. If a node is marked twice, this means that both adjacent parent elements connected to that node are to be coarsened. Only those nodes that are marked twice may be deleted. If a node is only marked once then that parent element is not allowed to be coarsened. For a parent element to be coarsened either all three of its nodes or two of its nodes must be deleted (allowing the 4:2 transition coarsening case). If a case is found that only one node is to be deleted, coarsening is not allowed.

This can be demonstrated algorithmically as follows :

DELELM(1..NELMS) = 1 if element is a parent to be deleted.

```
FOR I = 1 TO NELMS
  IF DELELM(I) = 1 THEN          {Mark all nodes connected to parent elements to be deleted}
     MARKER(ELEMENTS(I, 1)) = MARKER(ELEMENTS(I, 1)) + 1
     MARKER(ELEMENTS(I, 2)) = MARKER(ELEMENTS(I, 2)) + 1
```

```
    MARKER(ELEMENTS(I, 3)) = MARKER(ELEMENTS(I, 3)) + 1
  ENDIF
NEXT I
COUNT = 0
FOR I = 1 TO NELMS
  FOR J = 1 TO 3                    {Count no of nodes belonging to parent to be deleted}
    IF MARKER(ELEMENTS(I, J)) = 2 THEN
      COUNT = COUNT + 1
  NEXT J
  IF COUNT < 2 THEN          {Prevent coarsening}
    DELELM(I) = 0
  ENDIF
NEXT I
```

Due to a cross effect between elements, the above algorithm is performed as many times as necessary until a final list of elements to be deleted is obtained.

The above description is quite restrictive in coarsening the mesh. Therefore once a flow feature, typically a shock, has passed, a coarse mesh will not be recovered immediately. It would take several calls to this algorithm to recover the initial mesh.

4.6 Adaptive Refinement Tests

Two demonstrations of the adaptive refinement algorithm are performed, one for unsteady flow and the other for steady flow. The most important issue to be tested is the reduction in computational requirements over a uniform refinement case. It must also be seen whether the algorithm refines and coarsens the correct areas.

The computational time per time step is proportional to the number of elements in the mesh. An estimate of the savings in time and memory can be estimated by comparing the meshes with a uniform mesh with the same resolution. The adaptive refinement algorithm for transient problems is performed every five timeteps and takes approximately 40% as long as a single timestep, therefore the adaptive refinement algorithm itself increases the time of the entire simulation by 8%.

4.6.1 Shock interacting with a cylinder

In this test, a Mach 3 shock is modelled impinging on a cylinder.
Number of refinement levels = 3

Initial mesh = 1384 elements
Uniformly refined mesh = 1384. 4^3 = 88576 elements

Mesh 1 = 10902 elements Mesh 2 = 7150 elements
Mesh 3 = 10856 elements Mesh 4 = 15527 elements
Mesh 5 = 18060 elements

Average mesh resolution = [(10902 + 7150)/2 + (7150 + 10856)/2 + (10856 + 15527)/2 + (15527 + 18060)/2]/4 = 12004 elements

Time of simulation relative to time taken in uniform refinement case = 1.08(12004/88576) = 14.64%

The shock in the first mesh (fig4.6a) is smeared as the shock is defined on the initial coarse mesh. Once the mesh has been refined and a few timesteps have been performed, it steepens naturally.

The initial mesh must be fine enough to accurately approximate the boundary, in particular curves, as the approximation of the boundary will not increase with refinement. The fact that the cylinder is approximated by straight lines has a visible effect on the diffraction patterns. This effect can be reduced by increasing the resolution of the initial mesh and reducing the number of refinement levels.

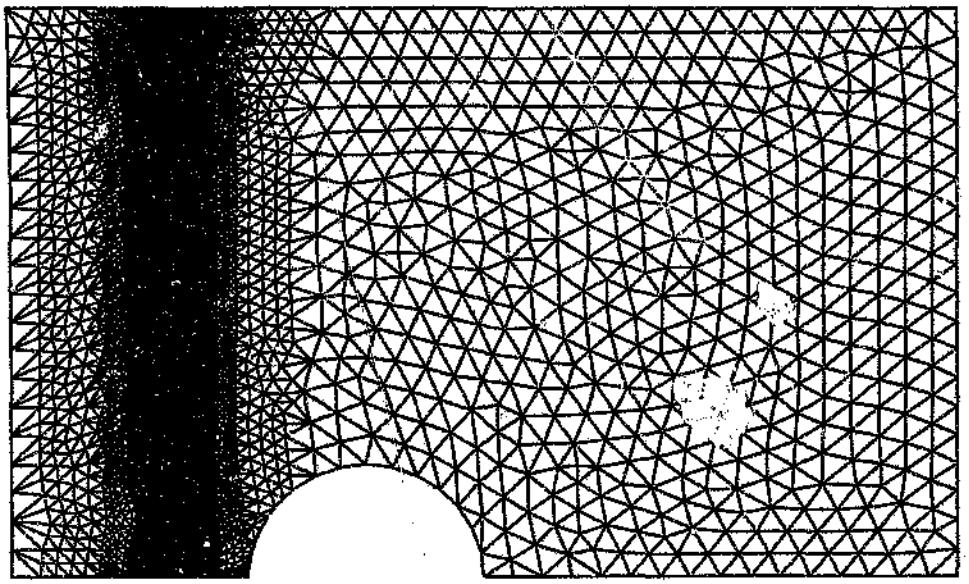
This problem took approximately six hours on an 80486 DX33 microprocessor. Simulations were also performed on a Convex C120 superminicomputer in scalar mode which ran approximately 4 times faster than the 80486 DX33 if no other users were on the machine.

4.6.2 Steady flow over a ramp

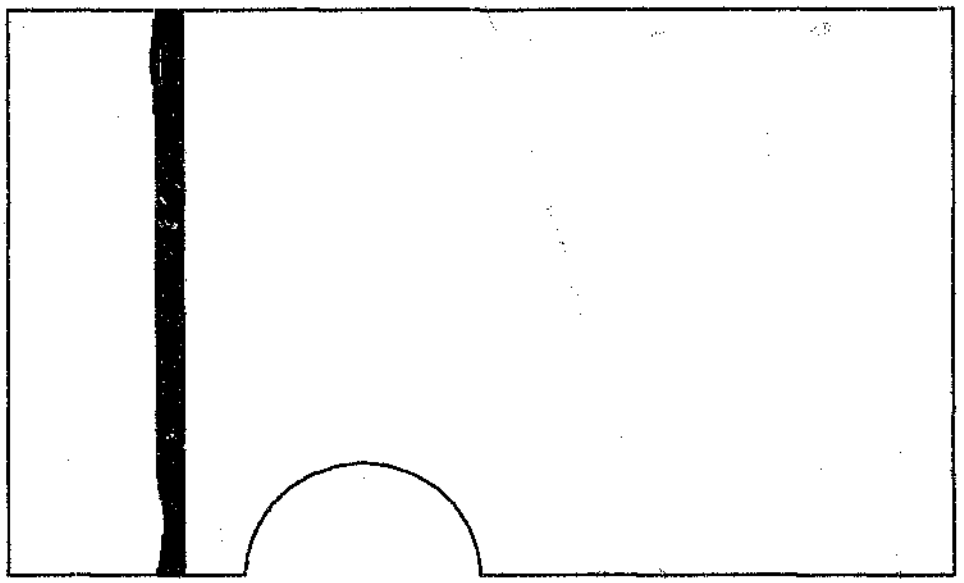
This test models steady Mach 3 flow in a contraction in a channel. Three levels of refinement are

performed. The uniform initial conditions are timestepped to steady state on the initial coarse mesh before the mesh is refined. The solution is converged to steady state again before the mesh is refined. This process is repeated until the desired refinement level is reached.

A similar performance increase over uniform meshes for the steady flow case as for the unsteady case is obtained. This problem took 2 hours 40 minutes to run on an 80486 DX33 microprocessor.

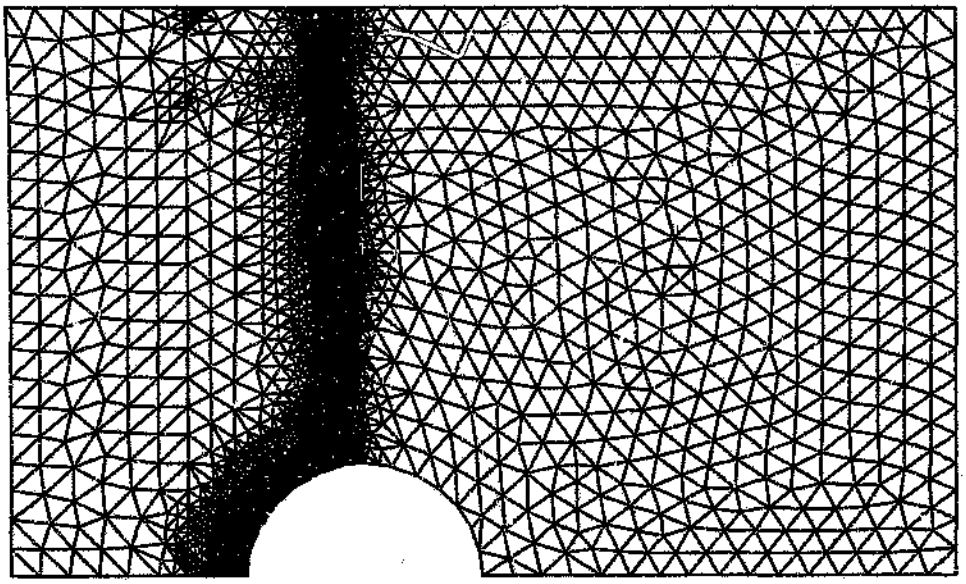


Mesh no 1, Elements = 10902

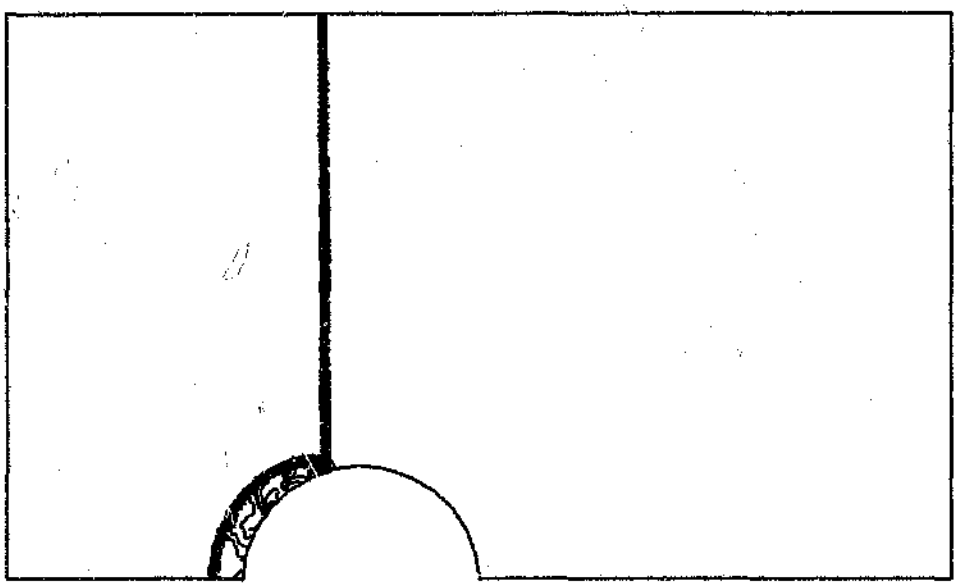


Density contours

Figure 4.6 (a) Time = 0

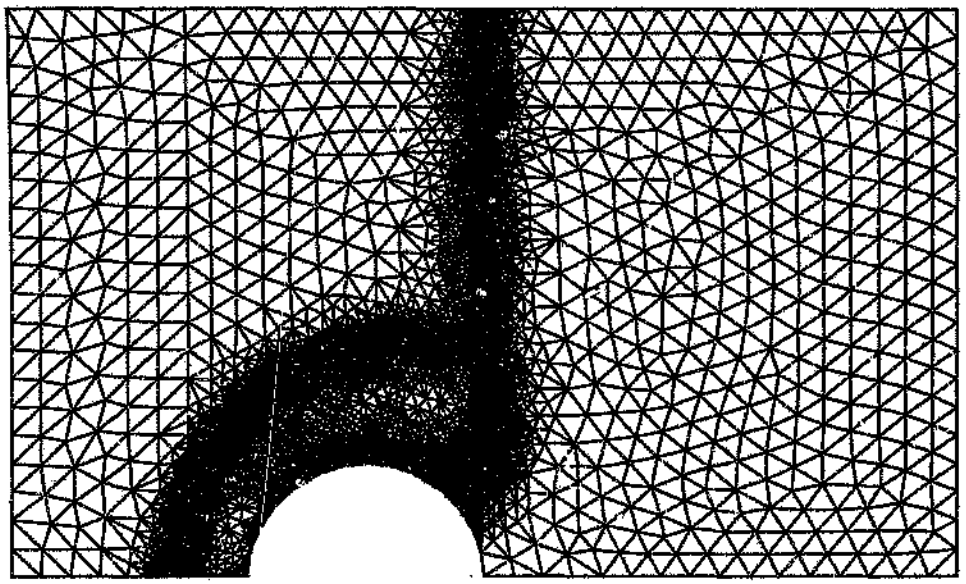


Mesh no 2, Elements = 7150

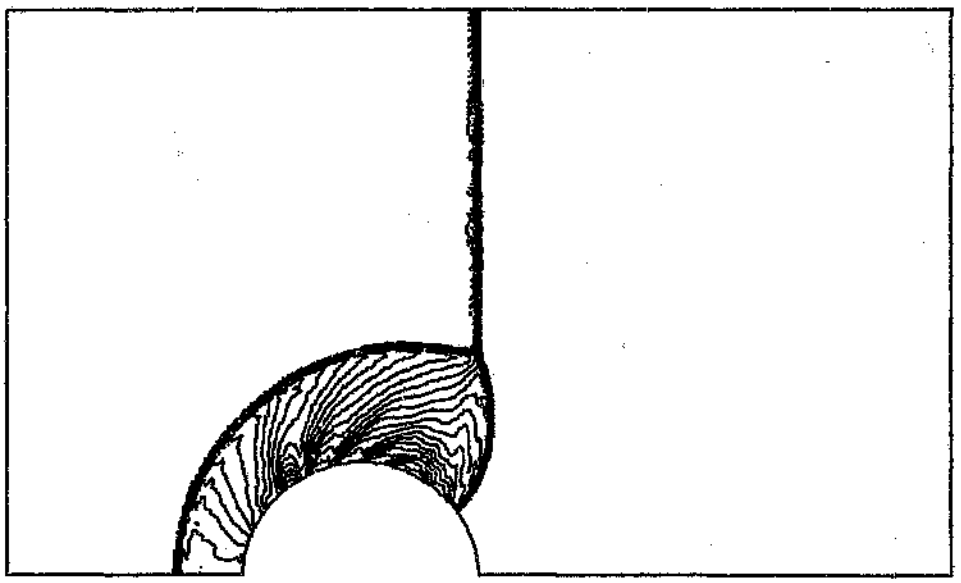


Density contours

Figure 4.6 (b) Time = 250 μs

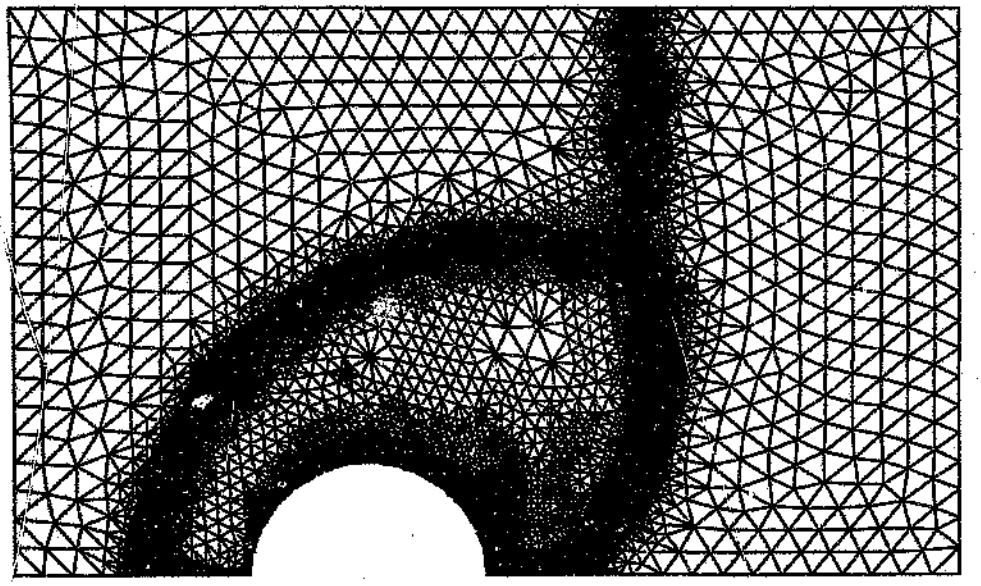


Mesh no 3, Elements = 10856

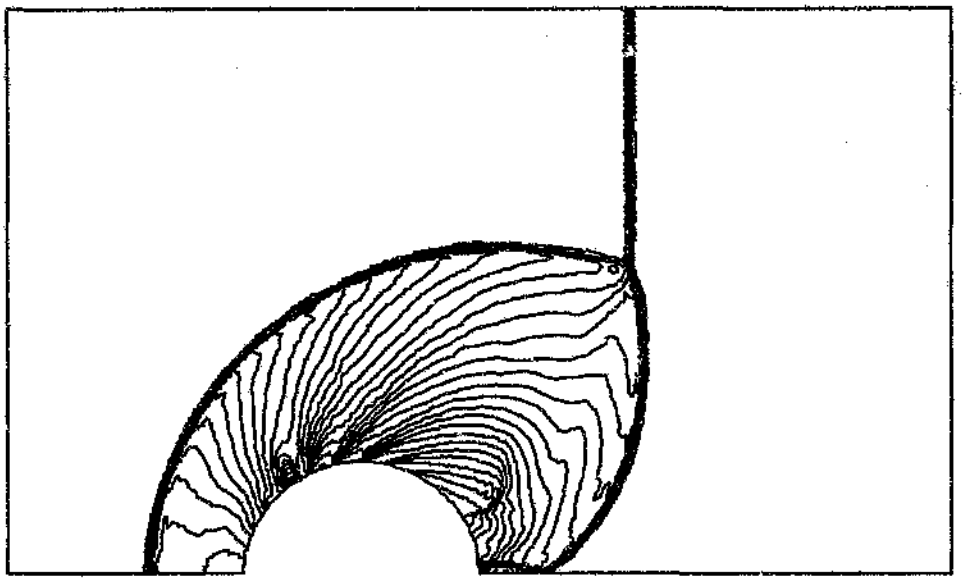


Density contours

Figure 4.6 (c) Time = 500 μs

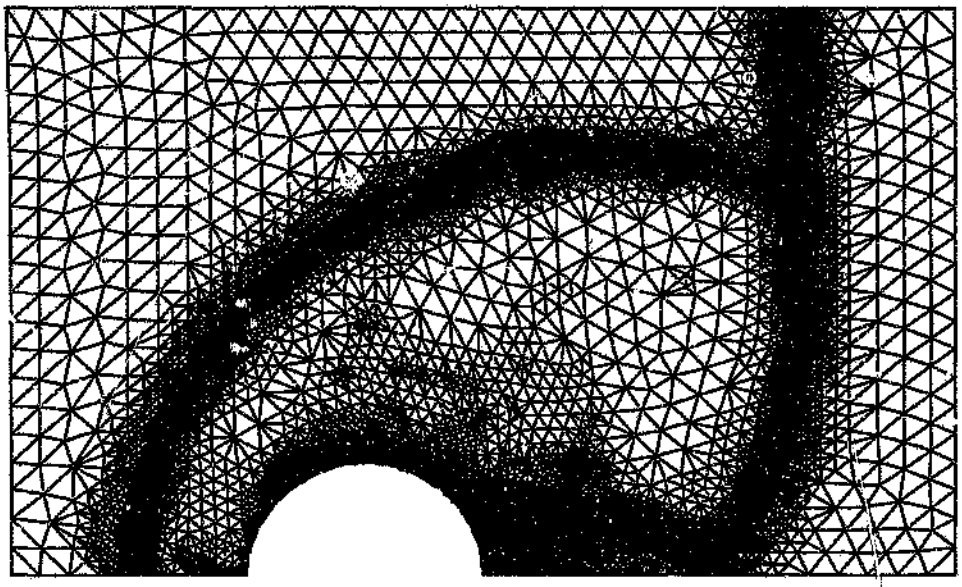


Mesh no 4, Elements = 15527

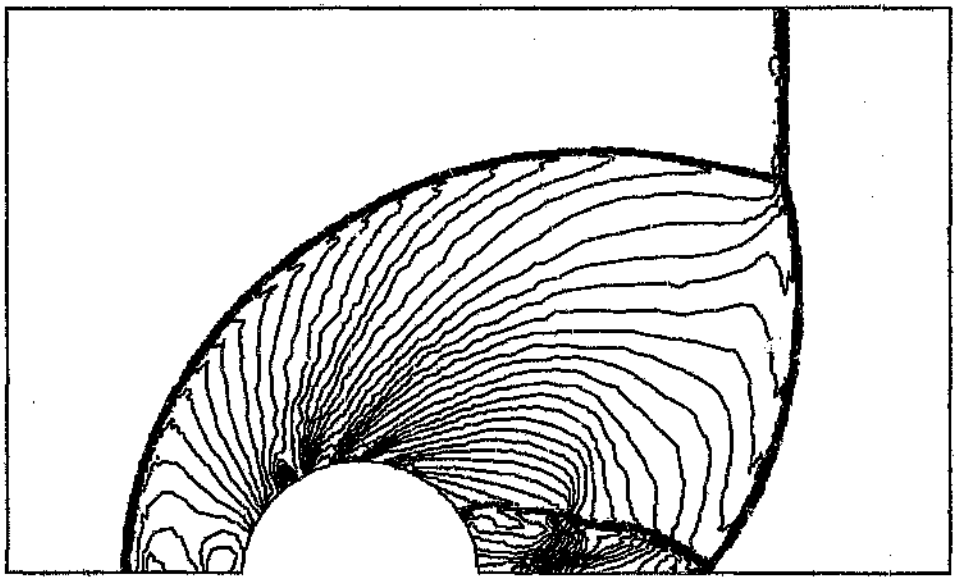


Density contours

Figure 4.6 (d) Time = 750 μs



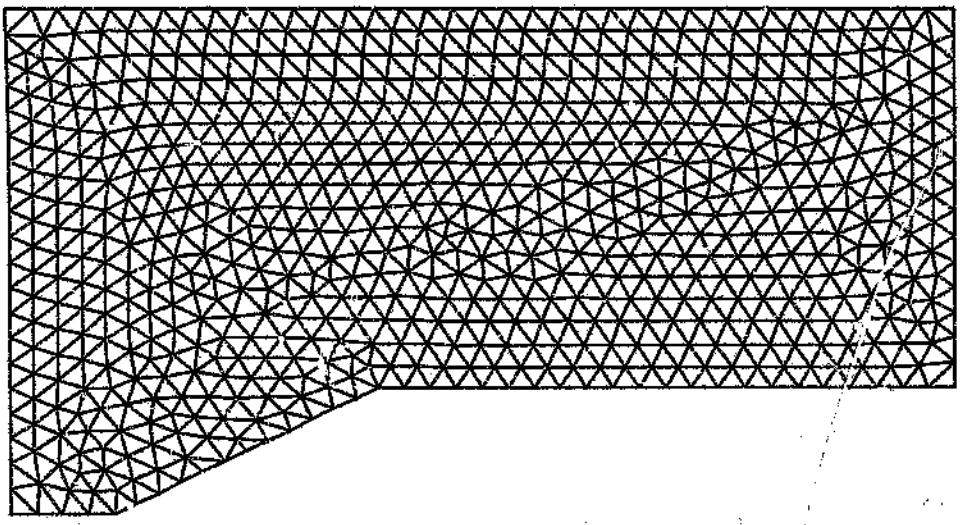
Mesh no 5, Elements = 18060



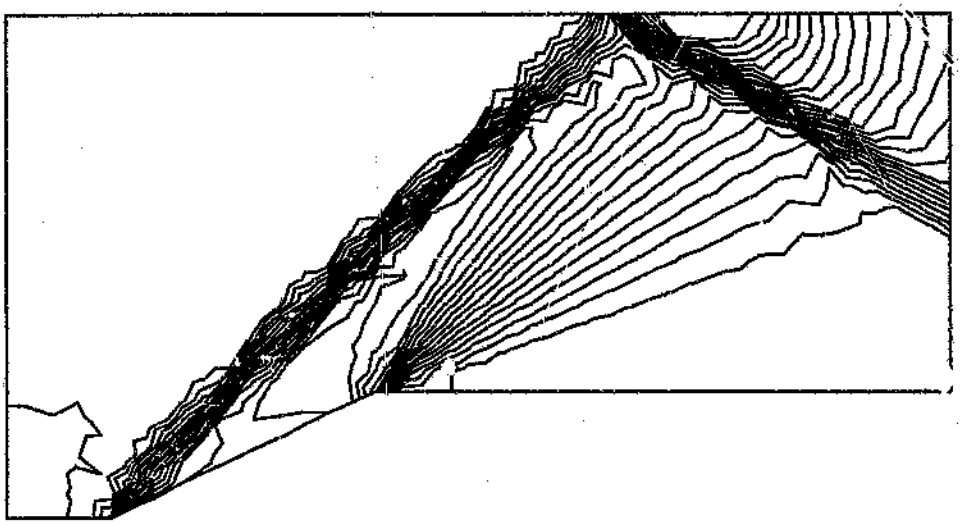
Density contours

Figure 4.6 (e) Time = 1000 μs

Figure 4.6 (a)-(e) Mach 3 Shock Interacting With a Cylinder

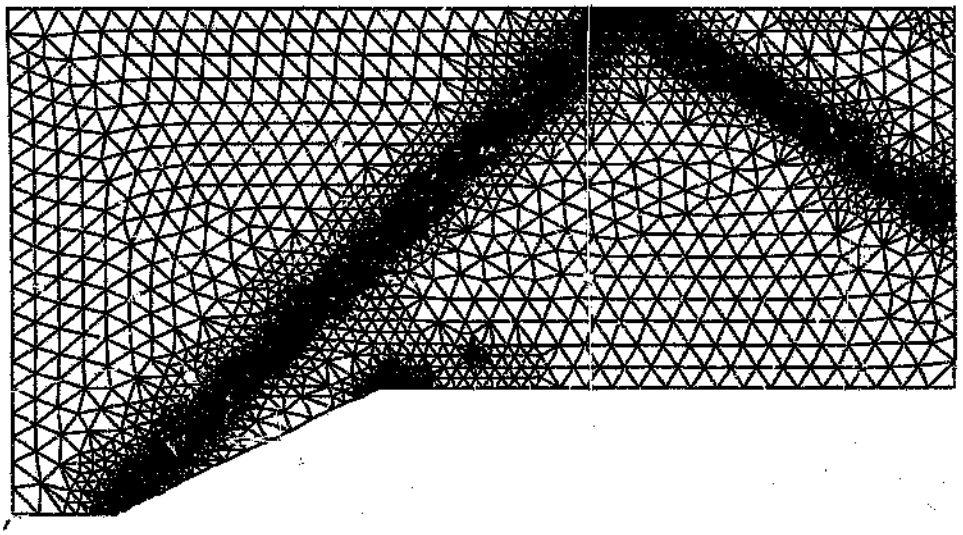


Mesh 1, Elements = 1343

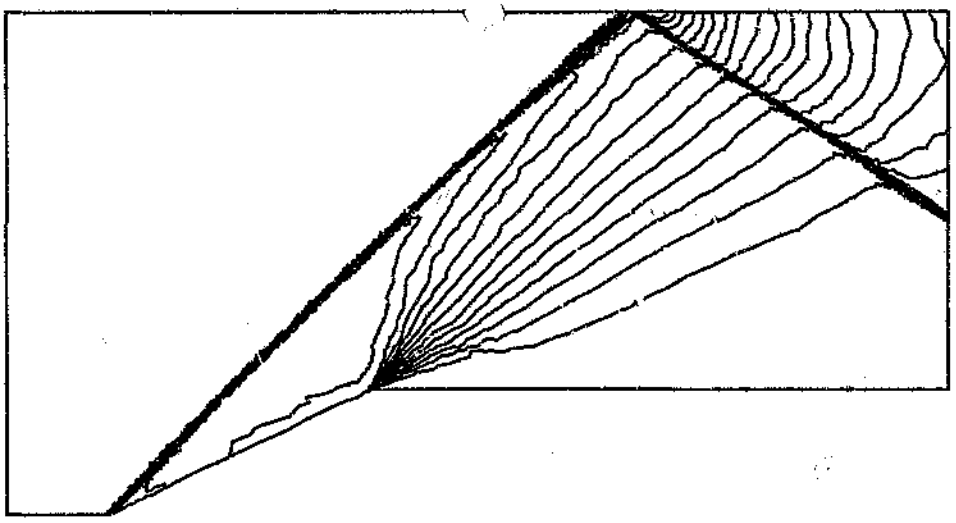


Density contours

Figure 4.7 (a) Converged solution on initial mesh



Mesh 4, Elements = 9387



Density contours

Figure 4.7 (b) Converged solution after 3 levels of refinement

Figure 4.7 (a)-(b) : Mach 3 flow over ramp in channel

5. COMPUTATIONAL RESULTS

5.1 Introduction

A computational fluid dynamics code is only as good as its ability to predict real world flow situations. In this chapter, several computational tests are performed and compared with analytical and experimental solutions. These test cases should provide an indication of how much confidence can be had in the solutions provided by this finite element program.

In any numerical scheme, two levels of approximations are made. The first approximation is the mathematical model used to represent the physical situation. In this case it is the compressible Euler equations that are used. The biggest limitation of this model is that it does not model viscous effects, however for the flows modelled here, viscous effects are generally considered to be negligible. This is especially the case for the transient problems involving a shock wave interacting with some geometrical feature, as the interactions occur over such a short time period that viscous features such as boundary layers and vortices have insufficient time to develop. The perfect gas approximation is also one which would start to break down at higher pressures and temperature. For the Tait equation, an isothermal assumption is made which again is only an approximation.

The second level of approximation is the accuracy to which the numerical scheme solves the mathematical model. The most obvious example of shortcomings in this case is the fact that discontinuities in the hyperbolic equation system are modelled as steep gradients. The fact that this is a high resolution code with an adaptive refinement scheme should minimise this problem though.

Wherever discrepancies are found it could result from errors in either level of approximation. It will be necessary to try and determine where discrepancies originate. The analytical solutions with which some results are compared, are based on the same inviscid flow theory on which the finite element solver is based. Therefore discrepancies between numerical solutions and analytical solutions will indicate an error in the way that the numerical scheme approximates the mathematical model used. When discrepancies occur between numerical and experimental results, the error could occur at either level of approximation. However if the code coincides with analytical solutions but does not correlate with experimental solutions, then it could be considered to be due to shortcomings in the mathematical model.

For the most part, fairly simple test cases are used. Analytical solutions are not available for the more complex cases. For these problems, only a couple of aspects of the flow are used for comparisons, in general, it is common to use various shock wave angles. Other possibilities are the use of pressure at various points in the flow.

A variety of flows are tested. Steady and unsteady compressible flows in air, two dimensional, three dimensional and axisymmetric geometries are modelled. But most of the emphasis is on two dimensional geometries with only one three dimensional model being tested. This is due to the fact that the three dimensional code has not reached an advanced stage of development.

An issue that must be considered is the validity of fairly simple test cases in demonstrating a codes ability, that is, whether a code that accurately models simple solutions will also give accurate complex solutions. Firstly, even though a geometry may be considered as simple, an unstructured mesh is unable to make use of that simplicity (which may well be an argument in favour of structured meshes for simple geometries). For the finite element algorithm, every geometry is a collection of arbitrarily oriented triangular or tetrahedral elements. The global solution is simply an assemblage of the solutions obtained from each element. Therefore if the solution within each element is valid and if the global solution is assembled correctly, the whole solution should be valid regardless of the complexity of the flow geometry or situation. In other words, the solver does not perceive situations to be complex or simple, whether the flow is simple or not, the same set of calculations have to be performed. It is therefore reasonable to assume that if the solution can be proven as valid for simple flow geometries, it can be assumed valid for complex geometries. It should also be noted that some of the flow situations occurring in some of these simple test cases are actually fairly complex.

Besides the evaluation problems, a few computational tests of general flow problems are performed as a demonstration of the algorithm's ability. They are supplied to demonstrate the code's ability to model complex geometries which was a major criteria of this work. No experimental or analytical solutions are provided for these problems.

5.2 Wedge Reflections

In these tests, the interaction of incident shock waves and wedges are modelled. The results are compared with experimental tests performed by Ben Dor and Glass[39]. Three tests are performed, each producing a different type of wedge reflection. These are a regular reflection, a simple Mach reflection and a complex Mach reflection. In their paper, Ben Dor and Glass advocate these experimental results as providing good test cases for numerical solutions.

These tests were performed in Argon. Argon is used in these tests because it maintains perfect gas behaviour for a greater range of pressures and temperatures than air. A low pressure (15 Torr) was used in the main test section. This has the advantage that overall prossures are reduced, to the benefit of the perfect gas assumption but this also makes it easier to achieve high Mach numbers in the shock tube.

Gas : Argon
$\gamma = 5/3 = 1.667$
$R = 208.1$
$T_0 = 300K$
$P_0 = 1996.5$ Pa (15 Torr)

All angles are measured to an accuracy of 0.2°.

Table 5.1 Computational and Experimental Results For Shock Wedge Interactions

θ	M	Experimental		Computational	
		Ψ	χ	Ψ	χ
60°	2.03	125.5°	-	125.8°	-
20°	2.82	79.9°	35.6°	80.6°	35.5°
30°	5.29	112.2°	40.6°	107.5°	40.8°

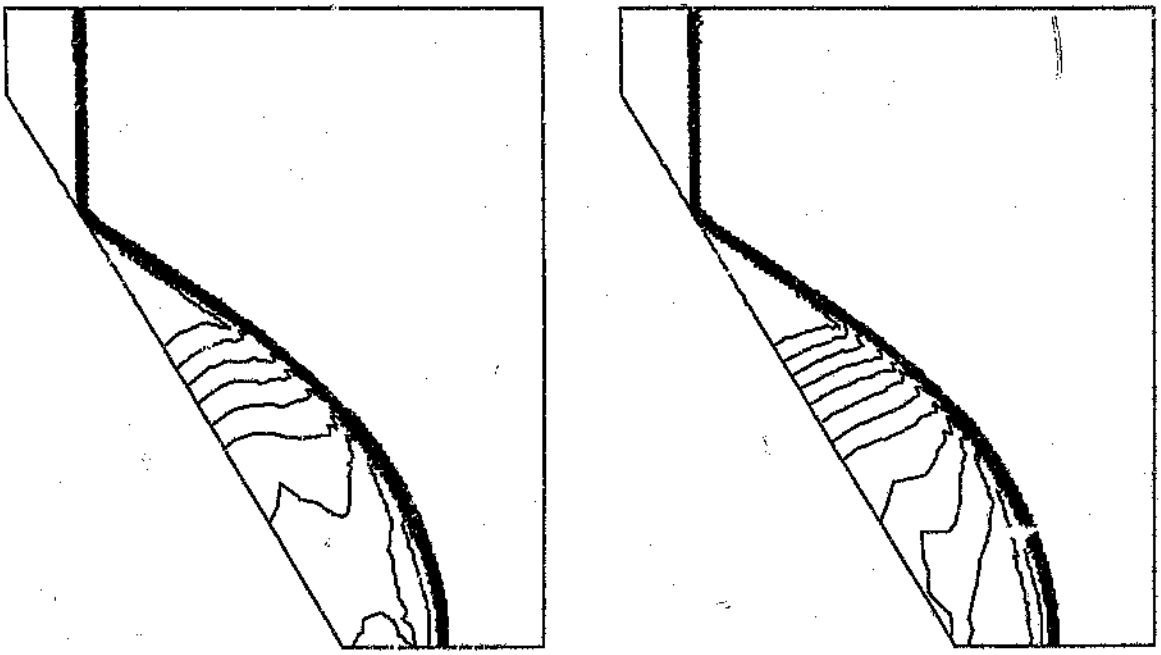
The first test (M= 2.03, $\theta = 60°$) models a regular reflection, the second test (M= 2.82, $\theta = 20°$) a simple Mach reflection and the third test (M= 5.29, $\theta = 30°$) a complex Mach reflection.

The second triple point trajectory angle (χ') for the complex Mach reflection problem was not used as a comparison because even though the complex Mach reflection phenomena is clearly observable, it is difficult to locate with any precision on either the computational or experimental solution the precise location of the 'kink' that is needed to determine the angle.

Though the geometries are simple, the flow situations arising, especially the complex Mach reflection are actually fairly complex and therefore provide fairly good test cases.

In general, the results correlate very well. It is only in the complex Mach reflection case that the angle of reflection showed some discrepancy with an error of 4.7°. As this is a fairly high Mach number test, it is quite likely that real gas effects have started to have an influence. This does seem to be the most probable cause as the lower Mach number tests gave good results. The surprising aspect was that the second triple point trajectory angle was still very accurately predicted. The numerical test did however correctly predict the complex Mach reflection configuration.

This problem does require further analysis which is not performed here. One possible approach is to model the problem using an alternative numerical scheme and to see whether the same discrepancy occurs. Another option would be to investigate the behaviour of Argon under these conditions and to attempt to use some form of real gas model to simulate the problem. However this is beyond the primary purpose of this work.



Density contours

Pressure contours

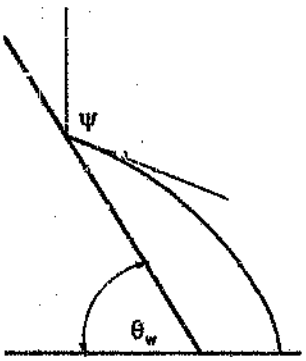
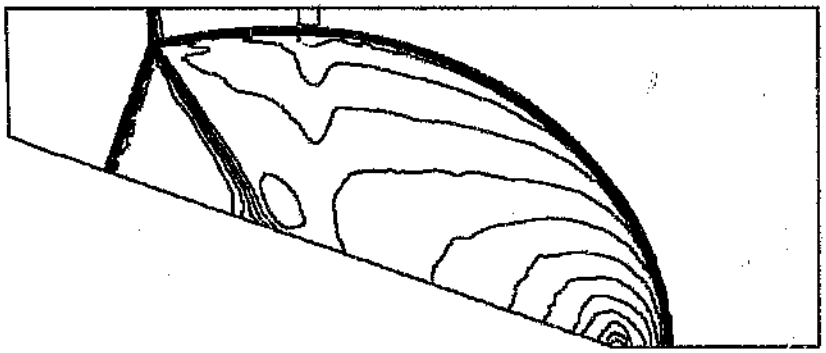
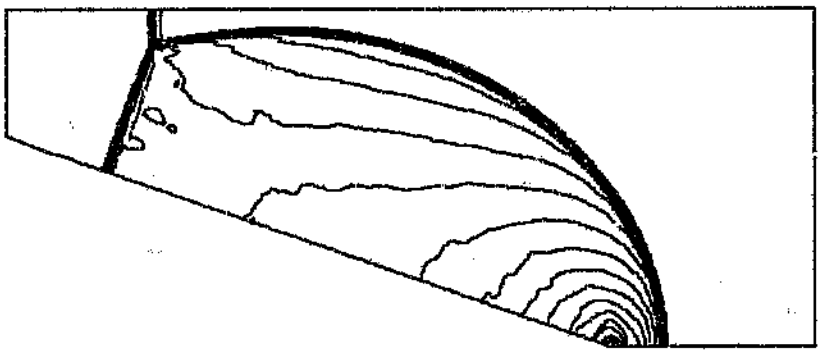


Figure 5.1 Regular reflection (M = 2.03, θ = 60°)



Density contours



Pressure contours

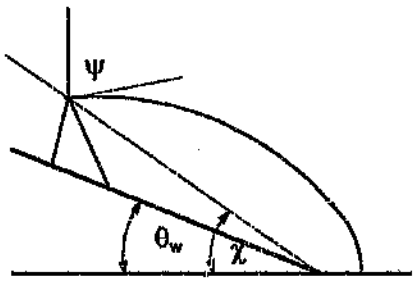
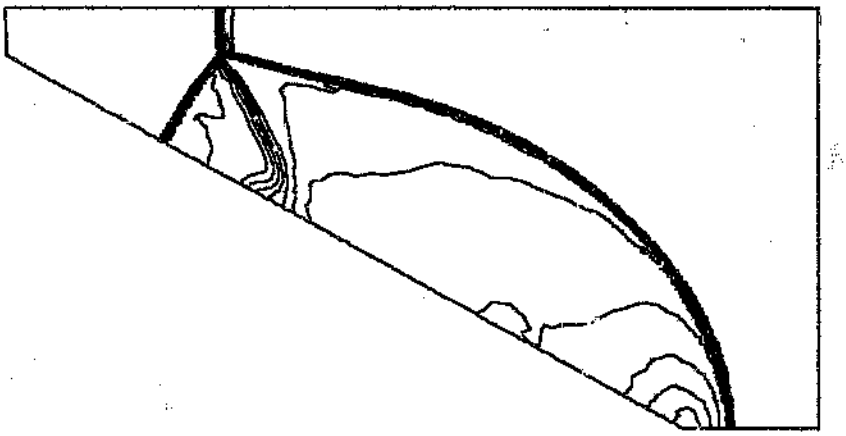
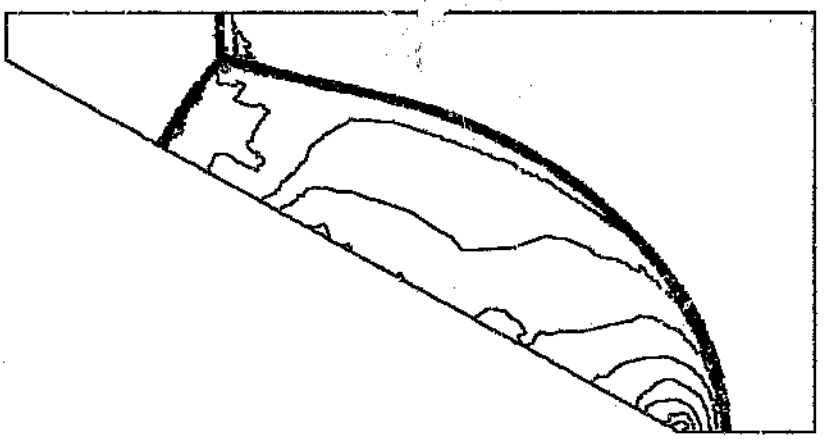


Figure 5.2 Simple Mach Reflection (M= 2.82, θ = 20°)



Density contours



Pressure contours

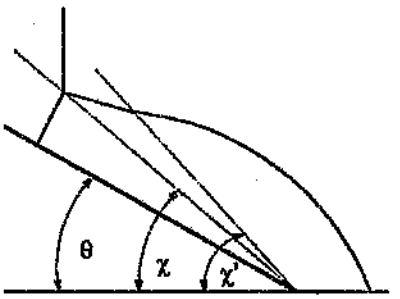


Figure 5.3 Complex Mach Reflection (M = 5.29, θ = 30°)

5.3 Steady Supersonic Flow over Wedges and Cones

Modelling steady flows over wedges and cones are convenient test cases for which very good analytical solutions exist. The parameter used as an evaluation is the wave angle. Two tests are performed for wedges and two for cones, with the same angles and flow conditions for the wedges and cones. These are very simple flow situations. As these are compared with analytical results, with the same inviscid, perfect gas assumptions, ideally an exact solution should be obtained from the simulation. The analytical solution for the wedges was calculated from White[40] and the analytical solution for the cones were determined from tables in John[41].

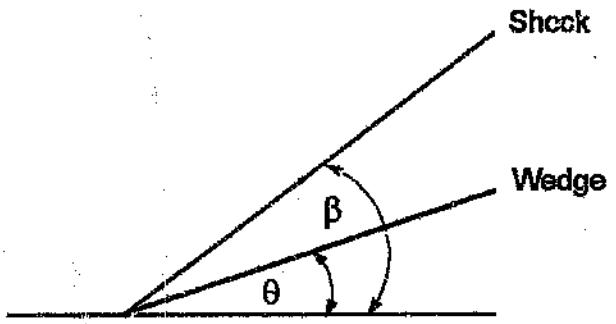


Figure 5.4 Schematic for steady flow over a wedge

Table 5.2 Analytical and Computational Results For Steady Flow Over Wedges and Cones

			Analytical	Computational
GEOMETRY	θ	**MACH NO**	β	β
WEDGE	22°	3	40.7°	40.4°
WEDGE	10°	2.5	32.6°	32.2°
CONE	22°	3	31.4°	31.4°
CONE	10°	2.5	25.4°	25.6°

Though the analytical and computational results are very close, for the wedge results the computational angle is consistently less than the analytical angle. This is likely to be due to the fact

the solution has not fully converged to steady state. A convergence tolerance of 0.5% was set for these tests. Tests run for poorer tolerances showed greater underprediction of wave angles.

This was tested by specifying a higher convergance tolerance for the cone problems. In this case the tolerance is set to 0.2%. The problem with underprediction is solved and the numerical errors are less than the measurement errors. However the cone solutions took more than twice the time to converge to steady state.

The reason that converging to the higher tolerance is so computationally expensive is due to the fact as the solution starts converging, the time scales involved get longer and longer but the timestep is still restricted by the stability criterion. In other words, a hundred timesteps could be performed with very little change in the solution. It is at this point that an implicit solver would be appropriate as very large timesteps could be performed. A possibility is for an explicit solver to be used until the solution converges to 2% during which time the solution changes fairly rapidly with time and then for an implicit solver to converge the solution further to the desired tolerance.

The reduced deflection for cones as compared to wedges for identical flow conditions and angles is immediately obvious. This is because the cone is less of an obstruction to the flow as it can go around the cone and expand radially.

These tests were run with differing resolutions. As the only feature of the flow is a single oblique shock, the resolution should have no effect on wave angle predicted.

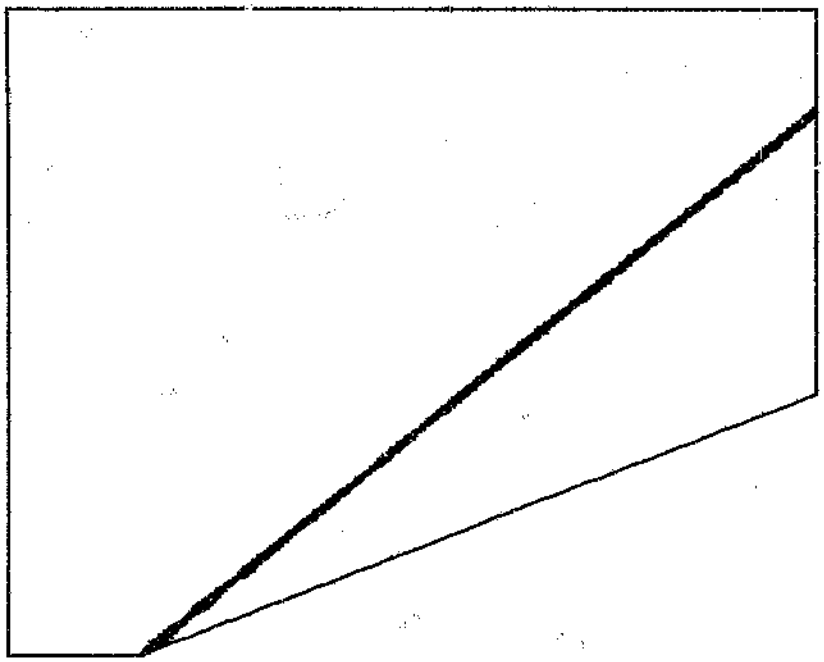


Figure 5.5 Steady flow over wedge - density contours (M = 3, θ = 22°)

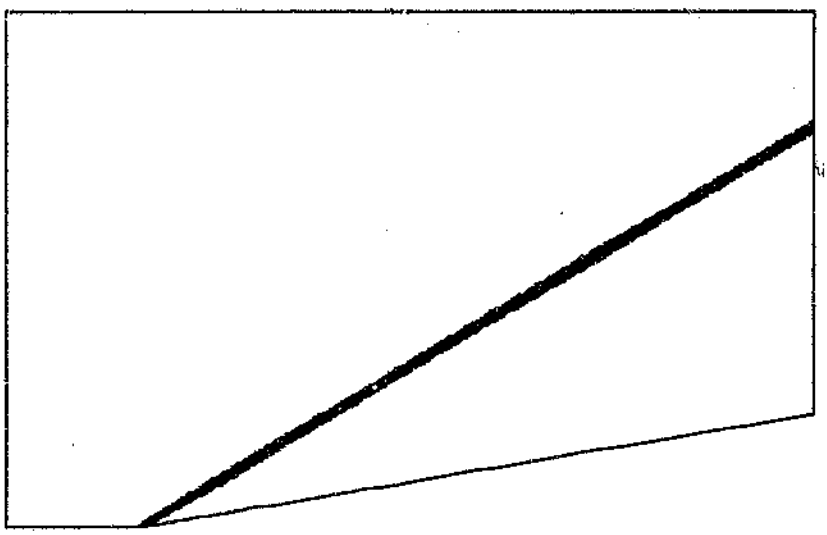


Figure 5.6 Steady flow over wedge - density contours (M = 2.5 θ = 10°)

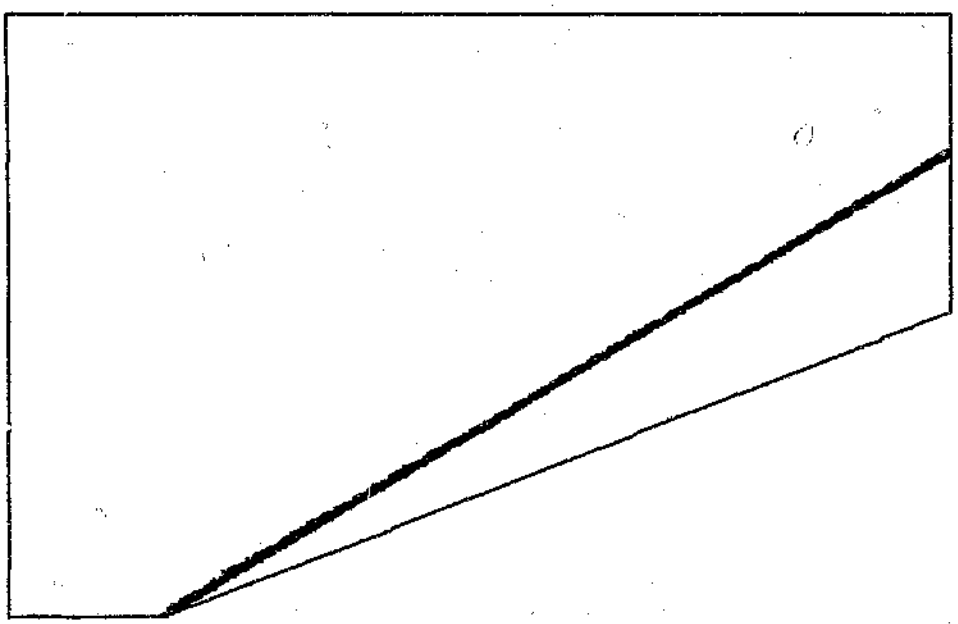


Figure 5.7 Steady flow over cone - density contours ($M = 3.0$ $\theta = 22°$)

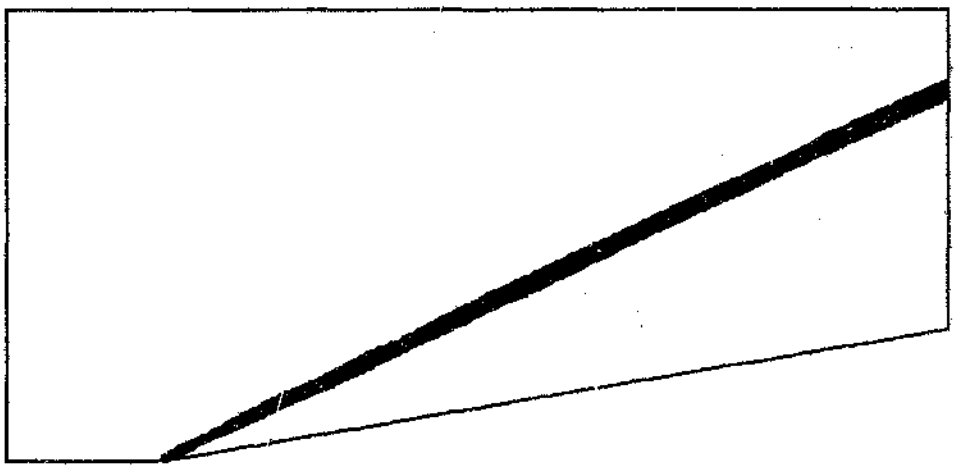


Figure 5.8 Steady flow over cone - density contours ($M = 2.5$, $\theta = 10°$)

5.4 Liquid Shock Waves

Liquid shock waves have not been extensively modelled numerically, in fact even experimentally the work at this stage has been limited. This is one of the few situations where a liquid would be modelled as being compressible. With a liquid shock, very high pressures are obtained though fluid velocities behind an incident shock are small.

Computational tests modelling shock waves in Fluorinert FC43 and water were performed. FC43 is a low sound speed liquid that is commonly used in transformers. The low sound speed makes it possible to generate very low speed shocks making it easier for transducers to make measurements and also for non-linear effects to become observable at lower Mach numbers.

With fluorinert FC43 as the medium two tests with different incident shock strengths were performed of regular reflections off a 45° wedge (figs 5.10 and 5.11). These tests are only compared with analytical results as appropriate experimental results were not available.

The Tait equation of state, described in sec 2.2.1, is used for these tests
Parameters for Tait equation for FC43:

n = 14.2
B = 567

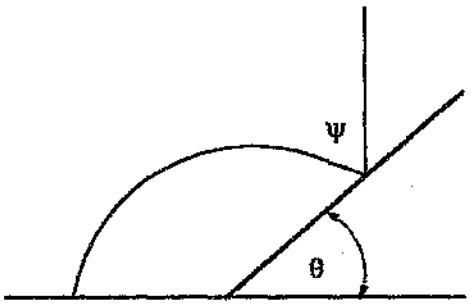


Figure 5.9 Liquid shock wave angles

Table 5.3 Analytical and Computational Results For Liquid Shock Wedge Reflections.

M_{shock}	P (bar)	θ	$\Psi_{analytical}$	$\Psi_{computational}$
1.046	100	45°	49.73°	50.1°
1.111	250	45°	58.72°	58.5°

Wha be noticed that even though post shock pressures are very high, the Mach numbers are very low. This is typical of liquid shock wave phenomena. For water, the Mach numbers for the same post shock pressures are even lower, M = 1.033 for 100 bar and M = 1.084 for 250 bar.

The comparisons are accurate to within the tolerance of measurement. As with the steady flow models discussed before, both the analytical solutions and computational solutions are based on the compressible Euler equations. If they did not match up, it would have to mean a mistake was made either analytically or numerically. It doesn't demonstrate the ability of the code to predict real fluorinert FC43 shocks. This would be dependant largely on the validity of the equation of state used.

A shock wave focussing problem in water, performed experimentally in Müller[2], is performed computationally and compared to the Schlieren photographs from Müller's experiment.

The equation of the parabola used to focus the shock wave is as follows :

$$x^2 = 4f(x + f) \qquad (f = 10mm)$$

where f is the focal distance. The diameter of the parabola is 56 mm but the diameter of the windows through which the Schlieren photographs were taken are only 46 mm in diameter. This must be taken into account when comparing the computational and experimental results. As the problem is symmetrical about the vertical axis, only one half of the geometry is modelled. The Mach number is 1.0006 which produces a post shock pressure of 7 bar.

The Tait equation parameters for water are :

B = 2955 bar
n = 7.44

The isopycnics are shown in figure 5.12 and the Schlieren photographs in fig 5.13. Unfortunately the times at which the photographs were taken did not match up exactly with the time intervals at which computational results were obtained but otherwise a good correlation is revealed.

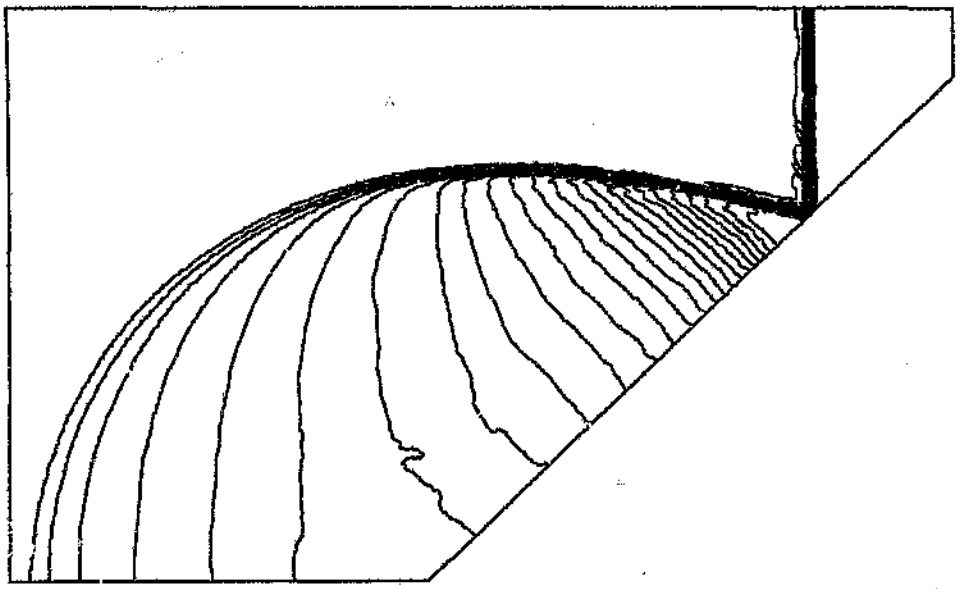


Figure 5.10 Liquid shock on wedge (M = 1.046, θ = 45°)

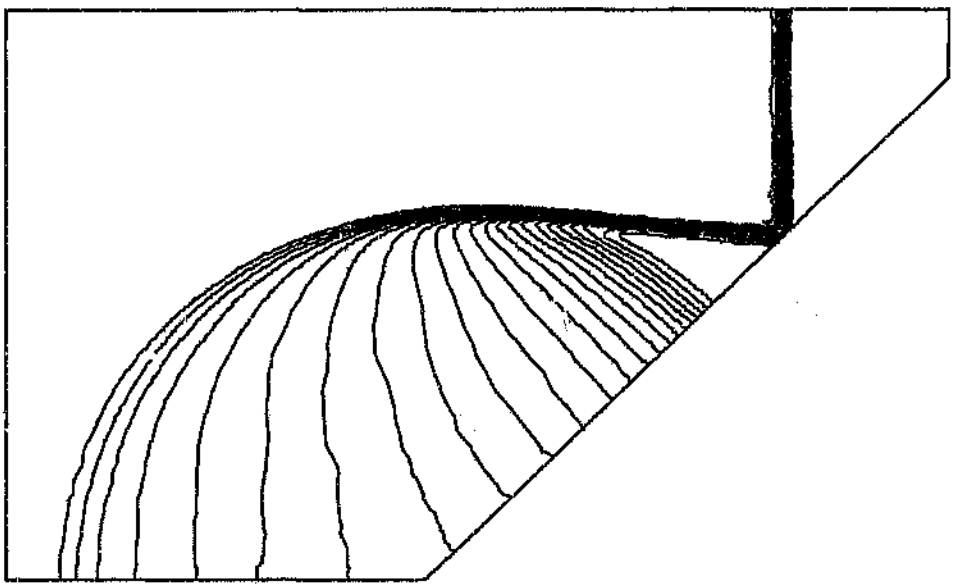


Figure 5.11 Liquid shock on wedge (M = 1.111, θ = 45°)

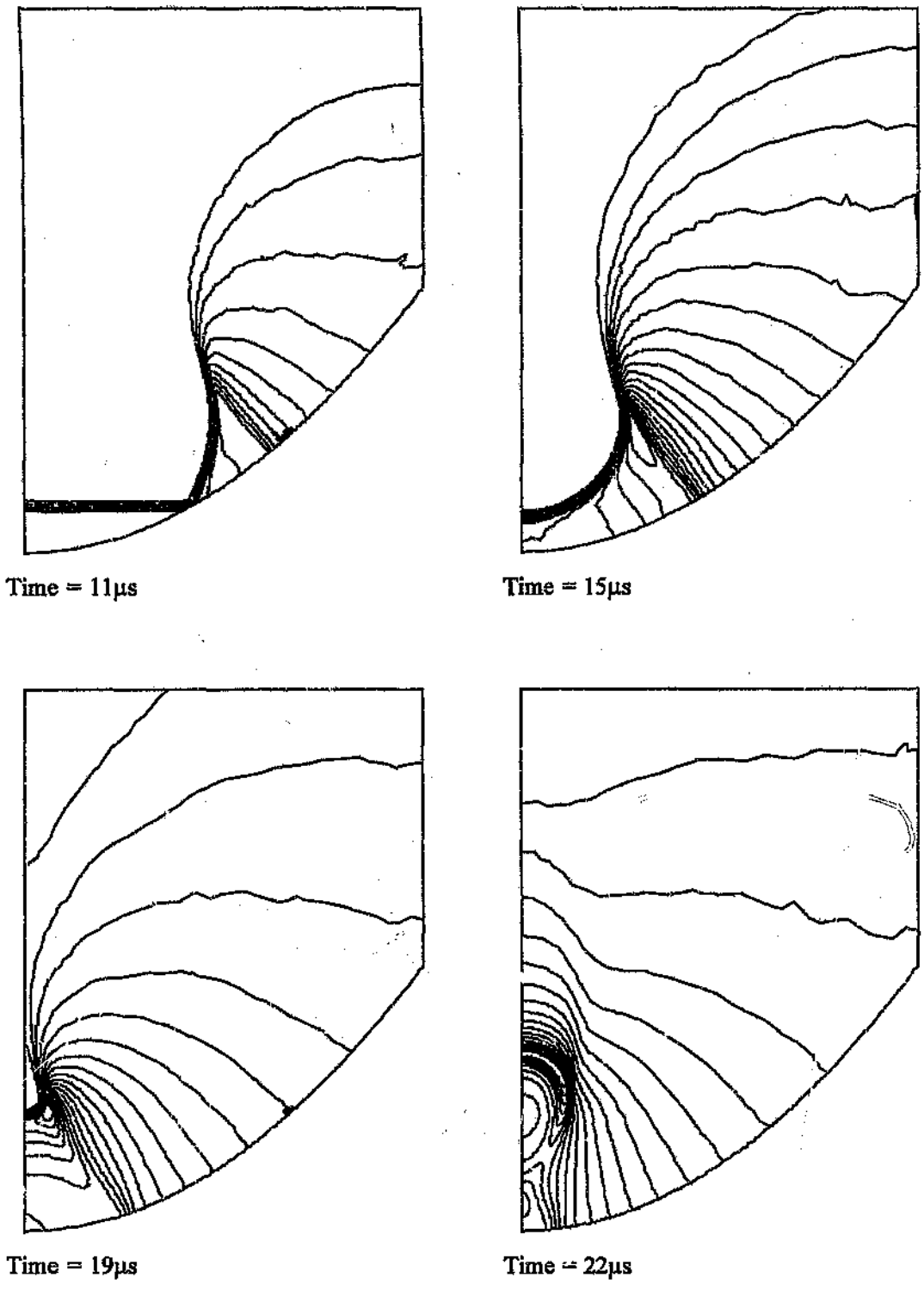


Figure 5.12 Shock wave focussing - computational isopycnics

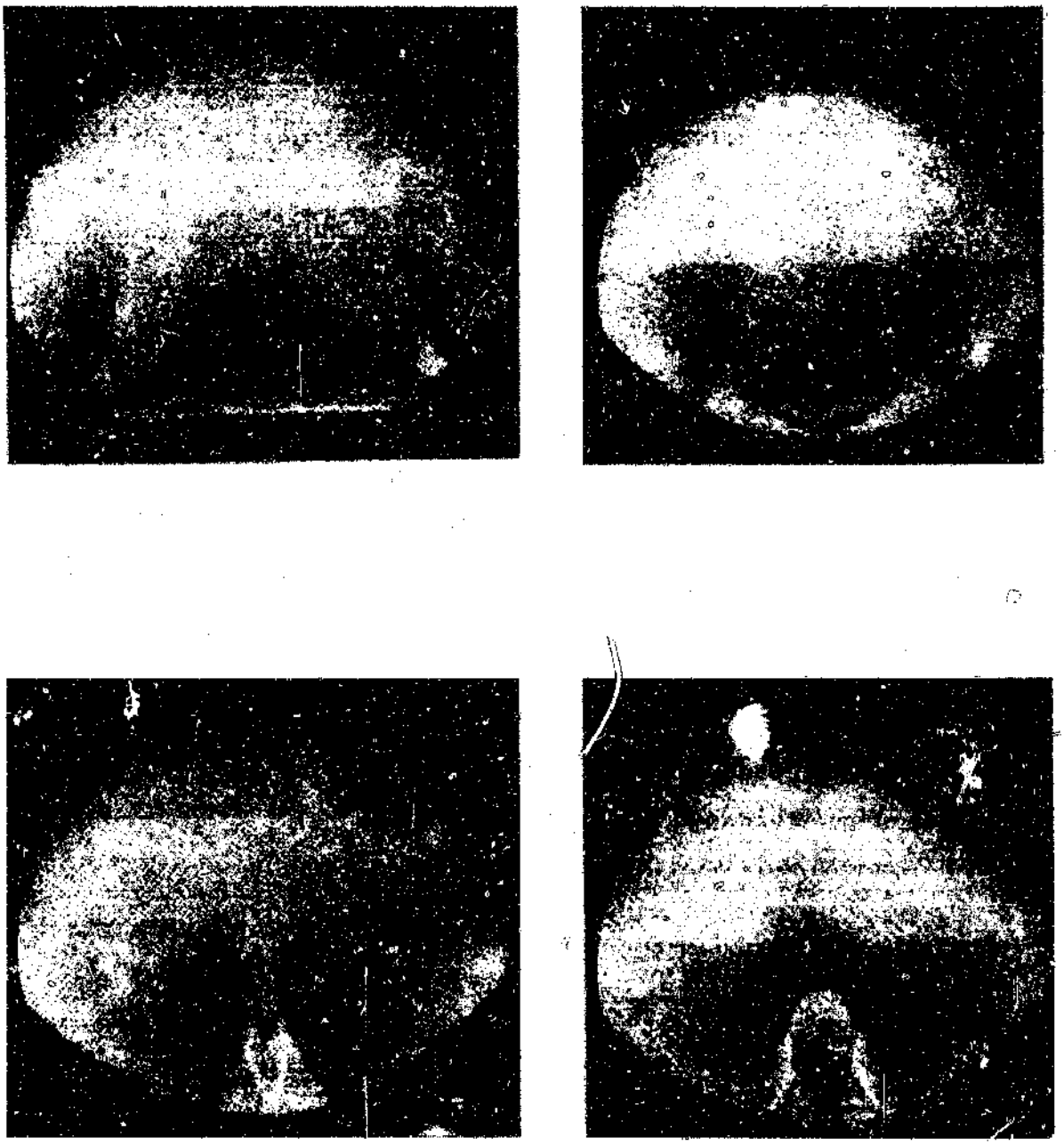


Figure 5.13 Shock wave focusing - Schlieren photographs

5.5 Three Dimensional Steady Supersonic Flow

A three dimensional version of the finite element code was implemented, but it was not refined to the same extent as the two dimensional code. This was due to the fact that a three dimensional implementation is considerably more complex than a two dimensional implementation. This is not so much due to the actual solver which extends readily to three dimensions but with the three dimensional geometry that is involved. Aspects like mesh generation, adaptive meshes and post processing is considerably more complex in three dimensions than in two dimensions.

A parametric mesh generator was implemented to generate the tetrahedral meshes. Three dimensional viewing algorithms were developed as well as a slicing algorithm which could display the contours on a plane slicing the geometry. These algorithms are described in Appendix B.

The geometry is that of a finite width 22 degree wedge in a supersonic wind tunnel. The wedge occupies half the width of the tunnel and the freestream velocity is Mach 3. Even though this is a three dimensional geometry, the flow on the wedge against the side wall is essentially two dimensional and becomes three dimensional as one moves away from the wall. The wave angle against the wall is used as a comparison.

The geometry and surface mesh is shown in fig. 5.14 and 5.15. In fig 5.16 the contours are displayed on a plane by the side wall against the wedge. This represents an essentially two dimensional flow situation. In fig 5.17 contours are displayed transverse to the wedge. These contours would be similar along the length of the wedge.

Computational = 40.4°
Analytical = 40.7°

The discrepancy is the same as that obtained in section 5.3 where the error was due to incomplete convergence. Though more comprehensive results for this flow situation was not available, the flow does appear in general to be physically correct. The expected vortex as the flow spills over the side of the wedge is clearly visible.

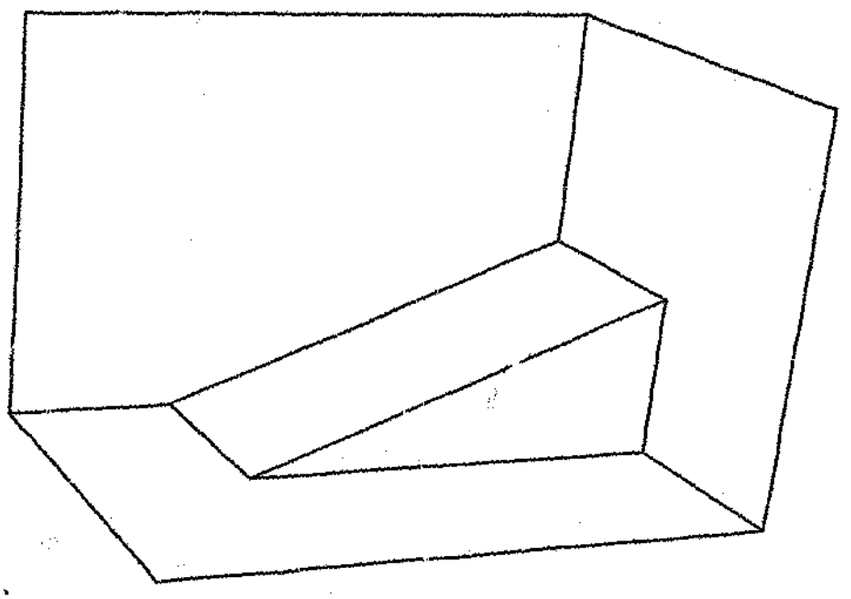


Figure 5.14 Three Dimensional Flow Geometry

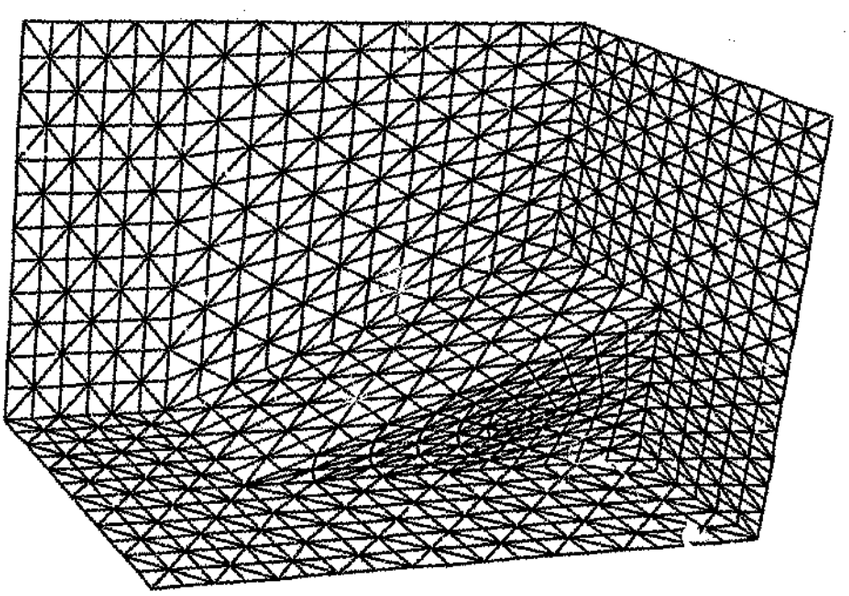


Figure 5.15 Surface Mesh

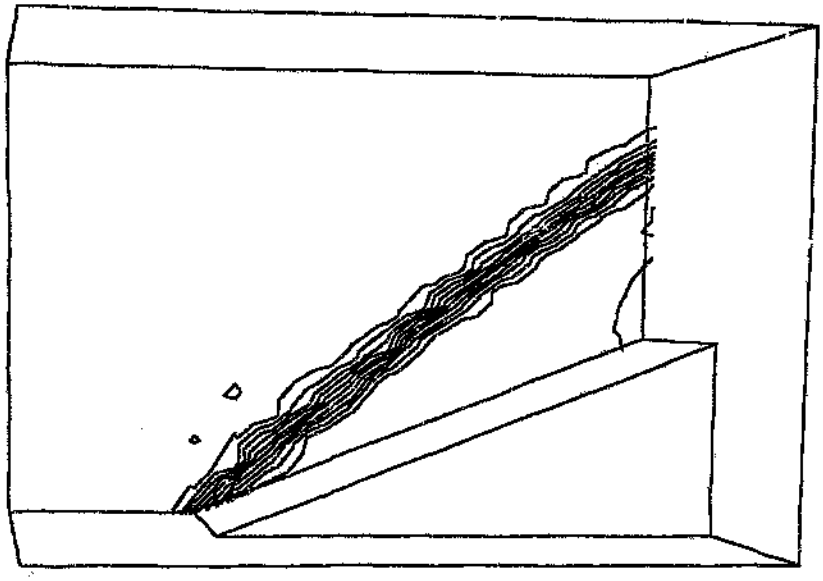


Figure 5.16 Density contours near rear wall

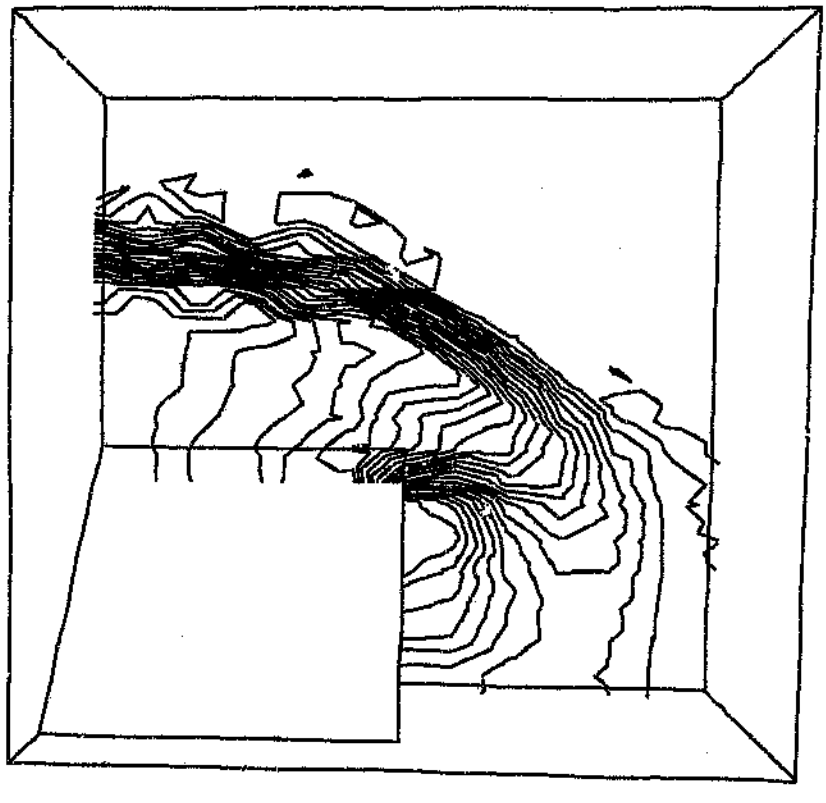


Figure 5.17 Density contours - transverse to wedge

5.6 Steady supersonic flow over a cylinder

A test is performed of steady supersonic flow over a cylinder. This is a symmetrical problem so only half of the cylinder is modelled. This is the same geometry as modelled in section 4.6 with an moving shock wave.

No analytical or experimental solution was obtained for this problem. This was to test the performance of the steady state solver in slightly more complex flows than that of the wedges and cones performed previously.

To demonstrate how the solution converges from uniform flow conditions, the solution is shown at various stages of convergence. The solution is considered to be converged when the maximum density change at all nodes over a time step is less than 0.5%. This will be referred to as the convergence tolerance. Convergence to steady state is only performed on the coarsest mesh after which the mesh is refined. It is clearly obvious how the initial convergence is fast and then proceeds to slow down.

This problem is one where the inviscid assumption clearly breaks down. It would be reasonable to expect that there will be flow separation and a vortex behind the cylinder. No such feature is predicted in the Euler model. It is reasonable to assume that whenever flow separation occurs, the validity of the inviscid assumption would start to break down. That is, if only the front half of the cylinder is being modelled, it would be acceptable to use an inviscid solution.

Clearly, for more general steady flow problems, it would be necessary to modify the solver for a full Navier-Stokes solution, and also to use implicit time integration.

5.7 Unsteady Flow Over Wedge

This is the same problem as in Sivier et al[42] modelling a shock wave impinging over and around a 55 degree wedge. This test is repeated here, see fig 5.19. This problem is interesting in that it has an inviscid vortex which is successfully predicted by this code, see fig 5.19 (b)-(c). Other features include a Mach stem and a slipline which are also successfully predicted, see fig 5.19(a)..

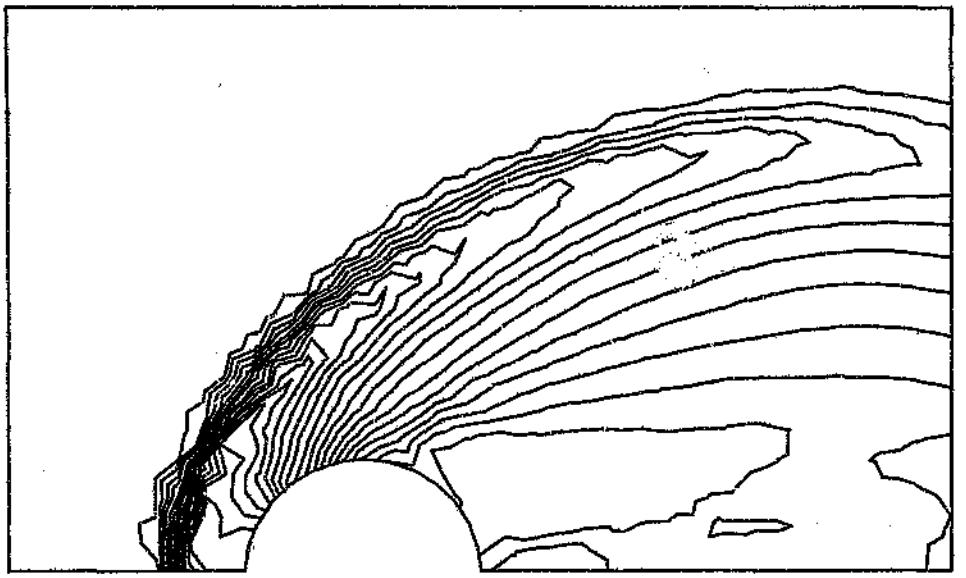


Figure 5.18(a) Convergance to max density change of 5%

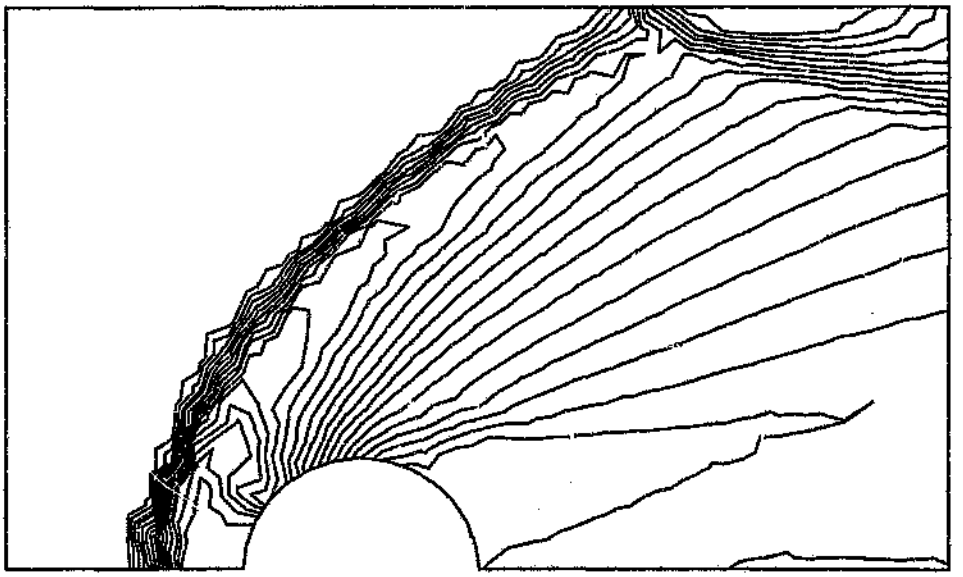


Figure 5.18(b) Convergance to max density change of 2%

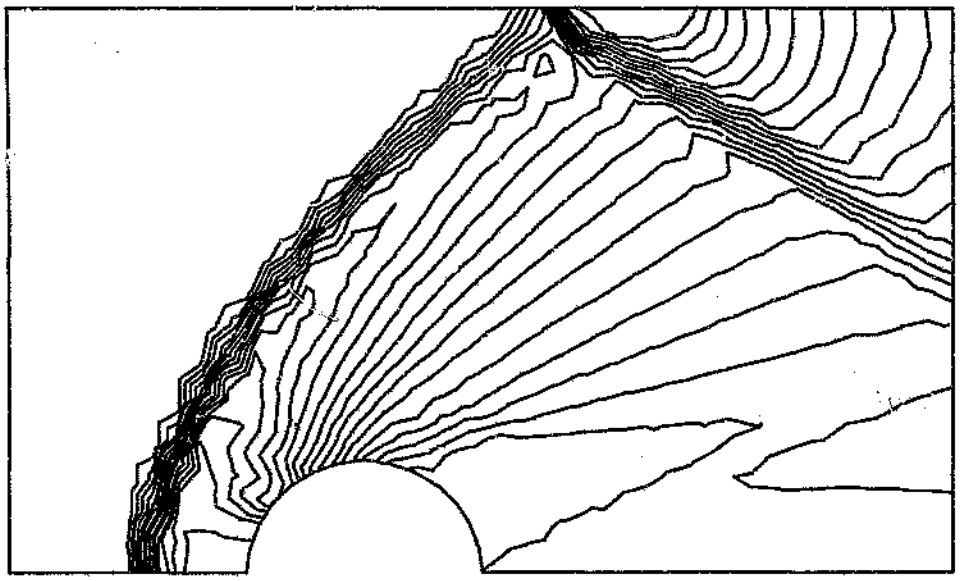


Figure 5.18(c) Convergance to max density change of 0.5%

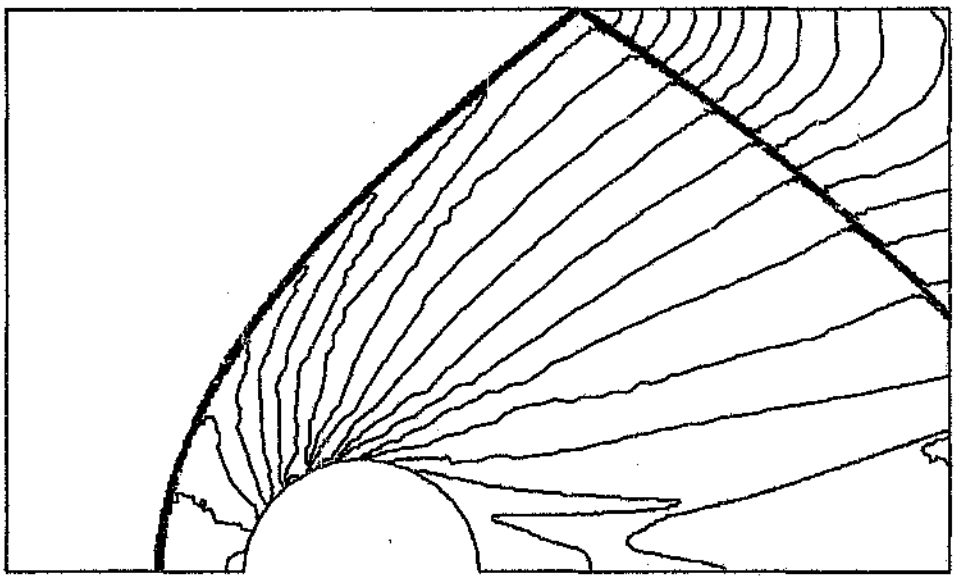
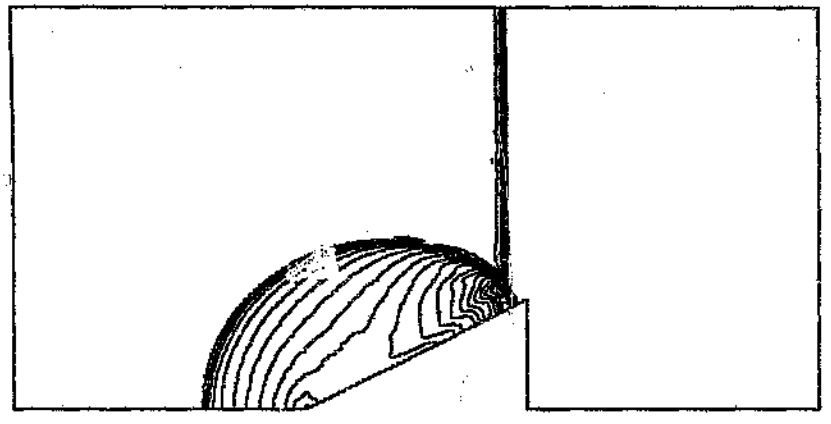
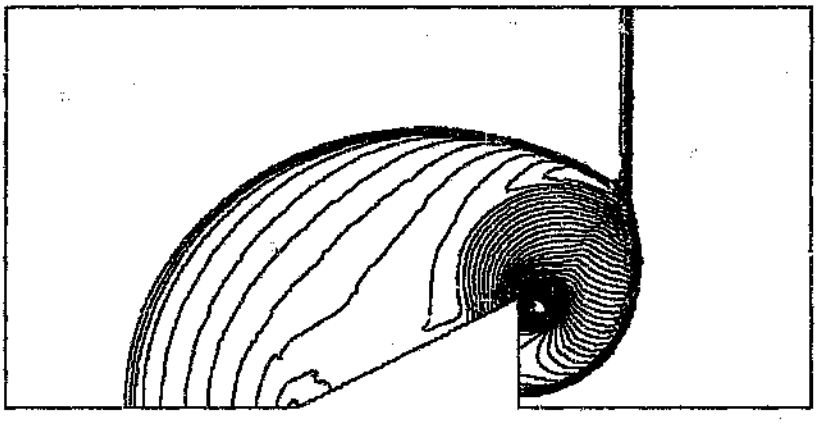


Figure 5.18(d) Converged solution after 3 levels of refinement

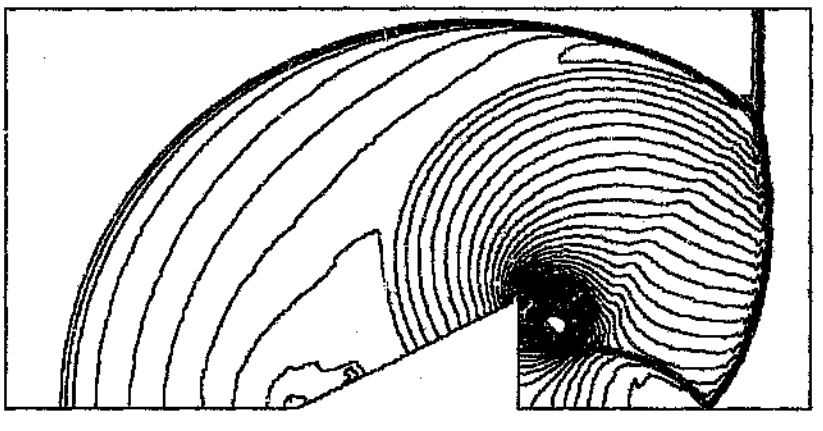
Figure 5.18(a)-(d) Steady flow over cylinder - isopycnics



(a)



(b)

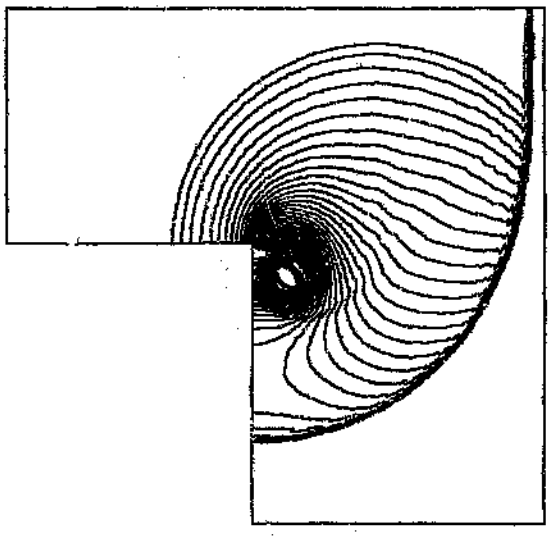


(c)

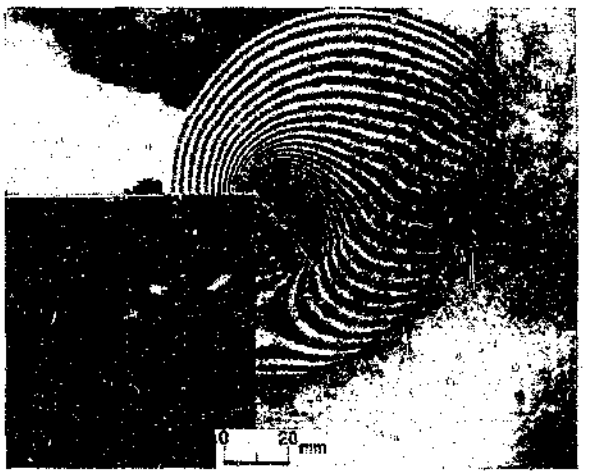
Figure 5.19 Shock over wedge - isopycnics

5.8 Shock diffraction around 90° corner

This is the classic test case of a M = 1.5 shock diffracting around a 90° corner. Computational results are compared with a laser interferogram[43] which is able to reveal density contours experimentally. A good correlation between computational and experimental results is demonstrated.



Computational isopycnics



Experimental isopycnics

Figure 5.20 Shock wave diffraction around 90° corner

6. CONCLUSIONS

A finite element algorithm was implemented for the modelling of general inviscid compressible flows. The following types of flows were modelled in this work : steady and unsteady two dimensional flows in air, steady axisymmetric flows in air, transient two dimensional flows in liquids and steady three dimensional flows in air.

An explicit Taylor-Galerkin solver was used to solve the compressible Euler equations. This was used in conjunction with a flux-corrected transport (FCT) algorithm which combines an oscillatory high order solution and a non oscillatory low order solution in such a manner that a high shock resolution is obtained without any non-physical oscillations and overshoots appearing in the solution. This algorithm was implemented for two and three dimensional flows. The Taylor-Galerkin algorithm was also extended to deal with axisymmetric flows which involved the addition of a source term to the hyperbolic conservation equations. The solver was used with the perfect gas equation for modelling flows in air and the Tait equation of state for modelling compressible liquids. This algorithm is able to provide time accurate transient solutions but is also used here for converging solutions to steady state.

An advancing front automatic mesh generator was developed to generate two dimensional uniform triangular meshes. This generator is able to handle complex connected domains with internal and arbitrarily curved boundaries. This algorithm enabled one to take full advantage of the unstructured nature of the finite element mesh, enabling complex unstructured meshes to be generated easily. This required the analyst to supply only the minimum information necessary to define the problem, avoiding the traditional approach of having to subdivide the geometry into blocks before it can be meshed..

An adaptive refinement algorithm was implemented which enabled an optimal mesh to be maintained without the intervention of the analyst. This algorithm refined the mesh where necessary enabling high gradient features to be accurately resolved and coarsening it again where necessary. This avoided the usual approach of having a uniformly refined mesh which is excessively fine over large areas of the domain. The adaptive refinement algorithm was able to reduce processor time and memory requirements by between 75 and 90% over solutions obtained using a uniform mesh, making adaptive meshing seem almost indispensable for these types of problems.

The combination of an automatic mesh generator and adaptive refinement algorithm enabled a black box type of approach to be taken with the solver. The analyst is able to describe the appropriate geometry, initial flow and boundary conditions, that is, the minium information necessary to define the problem, without having to be concerned with mesh details. This approach was found not only to be computationally efficient but dramatically reduced operator time as well. This approach is likely to become increasingly popular in the future.

The lack of an automatic mesh generator and an adaptive mesh algorithm for the three dimensional code proved to be its severest deficiency. Generating anything but the simplest meshes using traditional parametric mapping techniques proved to be difficult. The fact that the resultant mesh was non-optimal was also a problem especially as three dimensional problems require extensive computing resources. In fact without powerful meshing and adaptive meshing routines all the power of using an unstructured grid approach was effectively lost. Certainly as structured grid schemes are usually more efficient than unstructured grid schemes, using a structured grid in this case would have been advantageous. However, further development on the meshing and adaptive mesh techniques would change this situation and all the advantages experienced with these techniques with the two dimensional code could be achieved in the three dimensional case.

Generally, notwithstanding the above comments, unstructured grids was found to be an advantageous approach for general complex problems and further development would be needed in structured techniques for them to become competitive in the realm of general arbitrary geometries.

A number of verification tests were performed comparing numerical results with experimental and analytical results. Where comparisons were performed with analytical results, that is steady compressible flow over wedges and cones and liquid shock wave reflections, the results were very accurate. Wave angles were underpredicted very slightly due to lack of complete convergence in the steady state case where an implicit solver would have been more optimal enabling the solution to be converged further. It must be stated that this lack of accuracy is not excessive and it is still viable to use an explicit solver for steady flows. As analytical solutions and numerical solutions are based on the same idealisations this does demonstrate that the solver accurately approximates the compressible Euler model.

Comparisons of computational and experimental results were performed of regular, simple Mach and

complex Mach reflections on wedges. The angles for the regular reflection were predicted exactly. The poorest prediction was the angle of reflection for a complex Mach reflection. This discrepancy appears to be most likely due to real flow effects mainly due to the fact that the code accurately predicts situations where real flow effects are clearly absent. Further examination of this would be necessary, seeing when real effects become relevant and how to take them into account.

There are several logical extensions that can be performed on the solver. The most obvious is the solution of the full compressible Navier-Stokes equations which would not only include viscous effects but heat transfer and turbulence modelling. If such an extension was performed, the addition of an implicit solver would come in useful as in some situations, the Courant stability criterion could become excessively restrictive. Other useful extensions could include the modelling of moving boundaries through the use of arbitrary Langrangian Euler (ALE) implementations and the modelling of multiphase flows.

The science of modelling compressible flows has over the past few years approached a certain maturity where it can now be easily used as a practical tool rather than being a problem in itself. Many of the latest developments in compressible flow modelling were implemented in this work. While there is certainly scope for many more developments, the stated objective of developing a practical tool for modelling general inviscid compressible flows was achieved.

A wide variety of developments appear to be likely in the future. Domain decomposition methods appear to be becoming more popular though still do not match the flexibility of the finite element method. It is likely that research will continue in the use of higher order methods such as spectral methods and p type finite element meshes. Other possibilities is research into a unified approach where a single algorithm could be used for incompressible and compressible flows.

APPENDIX A

A1 Introduction

In this appendix all the integrations of the finite element equations are performed and the matrices are expanded fully. As the matrices for each degree of freedom are similar, all calculations are shown only for a single degree of freedom. Appropriate references describing the techniques used in this chapter are Stasa[37] and Zienkiewicz[30].

A2 The Linear Triangular Finite Element

The solution ϕ within each triangle is interpolated from the nodal solutions with the use of shape functions (N_i).

$$\phi = N_1\phi_1 + N_2\phi_2 + N_3\phi_3$$

The shape functions can be expressed in terms of real (x, y) and area coordinates(L_i). The area coordinates of a point p can be expressed as follows.

$$L_1 = \frac{A_{p23}}{A_{123}} \qquad L_2 = \frac{A_{p31}}{A_{123}} \qquad L_3 = \frac{A_{p12}}{A_{123}}$$

For a linear triangular finite element, the area coordinates (L_i) are equal to the shape functions (N_i). The shape functions are calculated as follows

$$N_1 = L_1 = (a_1 + b_1x + c_1y)$$
$$N_2 = L_2 = (a_2 + b_2x + c_2y)$$
$$N_3 = L_3 = (a_3 + b_3x + c_3y)$$

where

$$a_1 = (x_2y_3 - x_3y_2)/2A$$
$$b_1 = (y_2 - y_3)/2A$$
$$c_1 = (x_3 - x_2)/2A$$

$$a_2 = (x_3y_1 - x_1y_3)/2A$$
$$b_2 = (y_3 - y_1)/2A$$
$$c_2 = (x_1 - x_3)/2A$$

$$a_3 = (x_1y_2 - x_2y_1)/2A$$
$$b_3 = (y_1 - y_2)/2A$$
$$c_3 = (x_2 - x_1)/2A$$

where A, the area of the triangular element is

$$A = \frac{1}{2}\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix}$$

A3 The tetrahedral finite element

The solution is expressed in terms of shape functions as follows.

$$\phi = N_1\phi_1 + N_2\phi_2 + N_3\phi_3 + N_4\phi_4$$

where

$$N_1 = L_1 = (a_1 + b_1x + c_1y + d_1z)$$
$$N_2 = L_2 = (a_2 + b_2x + c_2y + d_2z)$$
$$N_3 = L_3 = (a_3 + b_3x + c_3y + d_3z)$$
$$N_4 = L_4 = (a_4 + b_4x + c_4y + d_4z)$$

where :

$$L_1 = \frac{V_{p234}}{V_{1234}} \qquad L_2 = \frac{V_{p341}}{V_{1234}} \qquad L_3 = \frac{V_{p412}}{V_{1234}} \qquad L_4 = \frac{V_{p123}}{V_{1234}}$$

and :

$$a_1 = \frac{1}{6V}\begin{vmatrix} x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{vmatrix} \qquad b_1 = -\frac{1}{6V}\begin{vmatrix} 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \\ 1 & y_4 & z_4 \end{vmatrix}$$

$$c_1 = -\frac{1}{6V}\begin{vmatrix} x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \\ x_4 & 1 & z_4 \end{vmatrix} \qquad d_1 = -\frac{1}{6V}\begin{vmatrix} x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ x_4 & y_4 & 1 \end{vmatrix}$$

The values at the other nodes are obtained by a cyclic rotation of the subscripts 1, 2, 3 and 4. The volume (V) of the tetrahedron is calculated as follows

$$V = \frac{1}{6}\begin{vmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{vmatrix}$$

A4 Integration Formulas

All integrations over linear triangular and tetrahedral elements can be calculated exactly, as opposed to numerically which is usual in finite element calculations using differently shaped elements and shape functions. Whenever integrations are to be performed, area and volume coordinates are used. Whenever a shape function is to be differentiated, cartesian coordinates are used.

The following integration formulas from Stasa[37] are used.

$$\int_l L_i^{\alpha} L_j^{\beta} dl = \frac{\alpha!\beta!}{(\alpha + \beta + 1)!} l$$

$$\int_A L_i^{\alpha} L_j^{\beta} L_k^{\gamma} dA = \frac{\alpha!\beta!\gamma!}{(\alpha + \beta + \gamma + 2)!} 2A$$

$$\int_V L_i^{\alpha} L_j^{\beta} L_k^{\gamma} L_m^{\delta} dV = \frac{\alpha!\beta!\gamma!\delta!}{(\alpha + \beta + \gamma + \delta + 3)!} 6V$$

A5 The Two Dimensional Finite Element Algorithm

First the increment for time n+1/2 is calculated from equation (15)

$$\delta U^{n+\frac{1}{2}} = -\frac{1}{2}\Delta t\left[\frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y}\right]$$

$$= -\frac{1}{2}\Delta t\left[b_1 F_{x1} + b_2 F_{x2} + b_3 F_{x3} + c_1 F_{y1} + c_2 F_{y2} + c_3 F_{y3}\right]$$

The integral over the element area (equation (14)) is

$$f_\Omega = \Delta t\int_\Omega F_x^{n+\frac{1}{2}} \frac{\partial N^T}{\partial x} d\Omega + \Delta t\int_\Omega F_y^{n+\frac{1}{2}} \frac{\partial N^T}{\partial y} d\Omega$$

$$= A\Delta t\frac{1}{3}\left((F_{x1} + F_{x2} + F_{x3})\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} + (F_{y1} + F_{y2} + F_{y3})\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}\right)$$

The boundary integral (equation (14)) is calculated as follows

$$f_\Gamma = -\Delta t\int_\Gamma F_x^{n+\frac{1}{2}} n_x N^T d\Gamma - \Delta t\int_\Gamma F_y^{n+\frac{1}{2}} n_y N^T d\Gamma$$

expanding :

$$f_T = -\Delta t n_x\int_\Gamma \begin{bmatrix} N_1(N_1F_{x1} + N_2F_{x2} + N_3F_{x3}) \\ N_2(N_1F_{x1} + N_2F_{x2} + N_3F_{x3}) \\ N_3(N_1F_{x1} + N_2F_{x2} + N_3F_{x3}) \end{bmatrix} d\Gamma - \Delta t n_y\int_\Gamma \begin{bmatrix} N_1(N_1F_{y1} + N_2F_{y2} + N_3F_{y3}) \\ N_2(N_1F_{y1} + N_2F_{y2} + N_3F_{y3}) \\ N_3(N_1F_{y1} + N_2F_{y2} + N_3F_{y3}) \end{bmatrix} d\Gamma$$

and integrating :

$$f_T = -\Delta tn_x \begin{bmatrix} \frac{1}{3}F_{x1} + \frac{1}{6}F_{x2} + \frac{1}{6}F_{x3} \\ \frac{1}{3}F_{x1} + \frac{1}{6}F_{x2} + \frac{1}{6}F_{x3} \\ \frac{1}{3}F_{x1} + \frac{1}{6}F_{x2} + \frac{1}{6}F_{x3} \end{bmatrix} - \Delta tn_y \begin{bmatrix} \frac{1}{3}F_{y1} + \frac{1}{6}F_{y2} + \frac{1}{6}F_{y3} \\ \frac{1}{3}F_{y1} + \frac{1}{6}F_{y2} + \frac{1}{6}F_{y3} \\ \frac{1}{3}F_{y1} + \frac{1}{6}F_{y2} + \frac{1}{6}F_{y3} \end{bmatrix}$$

The source term for axisymmetric flow (equation (20)) is

$$f_S = \int_\Omega S^{n+\frac{1}{2}} N^T d\Omega$$

expanding :

$$f_S = \int_\Omega \begin{bmatrix} N_1(N_1S_1 + N_2S_2 + N_3S_3) \\ N_2(N_1S_1 + N_2S_2 + N_3S_3) \\ N_3(N_1S_1 + N_2S_2 + N_3S_3) \end{bmatrix} d\Omega$$

and integrating :

$$f_S = A \begin{bmatrix} N_1\left(\frac{1}{6}S_1 + \frac{1}{12}S_2 + \frac{1}{12}S_3\right) \\ N_2\left(\frac{1}{12}S_1 + \frac{1}{6}S_2 + \frac{1}{12}S_3\right) \\ N_3\left(\frac{1}{12}S_1 + \frac{1}{12}S_2 + \frac{1}{6}S_3\right) \end{bmatrix}$$

For the solution of equation (19) the consistent mass matrix is needed

$$M_C = \int_\Omega N_j N_k d\Omega = \int_\Omega \begin{bmatrix} N_1N_1 & N_1N_2 & N_1N_3 \\ N_2N_1 & N_2N_2 & N_2N_3 \\ N_3N_1 & N_3N_2 & N_3N_3 \end{bmatrix} d\Omega = \frac{A}{12}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

The lumped matrix, obtained from summing up each row and placing the sum on the diagonal is as follows :

$$M_L = \frac{A}{3}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A6 The three dimensional finite element algorithm

The solution increment for time n+1/2 is calculated as follows

$$\delta U_e^{n+\frac{1}{2}} = -\frac{1}{2}\Delta t\left[\frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z}\right]$$

$$= -\frac{1}{2}\Delta t\left[b_i F_{xi} + c_i F_{yi} + d_i F_{zi}\right]$$

The three dimensional volume integral from equation (14) is :

$$f_\Omega = \Delta t\int_\Omega F_x^{n+\frac{1}{2}}\frac{\partial N^T}{\partial x}d\Omega + \Delta t\int_\Omega F_y^{n+\frac{1}{2}}\frac{\partial N^T}{\partial y}d\Omega + \Delta t\int_\Omega F_z^{n+\frac{1}{2}}\frac{\partial N^T}{\partial z}d\Omega$$

expanding and integrating :

$$f_\Omega = \frac{1}{4}\Delta t(F_{x1} + F_{x2} + F_{x3} + F_{x4})\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} + \frac{1}{4}\Delta t(F_{y1} + F_{y2} + F_{y3} + F_{y4})\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix}$$

$$+ \quad \frac{1}{4}\Delta t(F_{z1} + F_{z2} + F_{z3} + F_{z4})\begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix}$$

The three dimensional boundary integral is :

$$f_\Gamma = -\Delta t\int_\Gamma F_x^{n+\frac{1}{2}} n_x N^T d\Gamma - \Delta t\int_\Gamma F_y^{n+\frac{1}{2}} n_y N^T d\Gamma - \Delta t\int_\Gamma F_z^{n+\frac{1}{2}} n_z N^T d\Gamma$$

expanding :

$$f_\Gamma = -\Delta t n_x\int\begin{bmatrix} N_1(N_1F_{x1} + N_2F_{x2} + N_3F_{x3} + N_4F_{x4}) \\ N_2(N_1F_{x1} + N_2F_{x2} + N_3F_{x3} + N_4F_{x4}) \\ N_3(N_1F_{x1} + N_2F_{x2} + N_3F_{x3} + N_4F_{x4}) \\ N_4(N_1F_{x1} + N_2F_{x2} + N_3F_{x3} + N_4F_{x4}) \end{bmatrix}$$

$$-\Delta t n_y\int\begin{bmatrix} N_1(N_1F_{y1} + N_2F_{y2} + N_3F_{y3} + N_4F_{y4}) \\ N_2(N_1F_{y1} + N_2F_{y2} + N_3F_{y3} + N_4F_{y4}) \\ N_3(N_1F_{y1} + N_2F_{y2} + N_3F_{y3} + N_4F_{y4}) \\ N_4(N_1F_{y1} + N_2F_{y2} + N_3F_{y3} + N_4F_{y4}) \end{bmatrix}$$

$$-\Delta t n_z\int\begin{bmatrix} N_1(N_1F_{z1} + N_2F_{z2} + N_3F_{z3} + N_4F_{z4}) \\ N_2(N_1F_{z1} + N_2F_{z2} + N_3F_{z3} + N_4F_{z4}) \\ N_3(N_1F_{z1} + N_2F_{z2} + N_3F_{z3} + N_4F_{z4}) \\ N_4(N_1F_{z1} + N_2F_{z2} + N_3F_{z3} + N_4F_{z4}) \end{bmatrix}$$

integrating :

$$f_T = -\Delta t n_x A \begin{bmatrix} \frac{1}{6}F_{x1}^{n+\frac{1}{2}} + \frac{1}{12}F_{x2}^{n+\frac{1}{2}} + \frac{1}{12}F_{x3}^{n+\frac{1}{2}} + \frac{1}{12}F_{x4}^{n+\frac{1}{2}} \\ \frac{1}{6}F_{x1}^{n+\frac{1}{2}} + \frac{1}{12}F_{x2}^{n+\frac{1}{2}} + \frac{1}{12}F_{x3}^{n+\frac{1}{2}} + \frac{1}{12}F_{x4}^{n+\frac{1}{2}} \\ \frac{1}{6}F_{x1}^{n+\frac{1}{2}} + \frac{1}{12}F_{x2}^{n+\frac{1}{2}} + \frac{1}{12}F_{x3}^{n+\frac{1}{2}} + \frac{1}{12}F_{x4}^{n+\frac{1}{2}} \\ \frac{1}{6}F_{x1}^{n+\frac{1}{2}} + \frac{1}{12}F_{x2}^{n+\frac{1}{2}} + \frac{1}{12}F_{x3}^{n+\frac{1}{2}} + \frac{1}{12}F_{x4}^{n+\frac{1}{2}} \end{bmatrix}$$

$$-\Delta t n_y A \begin{bmatrix} \frac{1}{6}F_{y1}^{n+\frac{1}{2}} + \frac{1}{12}F_{y2}^{n+\frac{1}{2}} + \frac{1}{12}F_{y3}^{n+\frac{1}{2}} + \frac{1}{12}F_{y4}^{n+\frac{1}{2}} \\ \frac{1}{6}F_{y1}^{n+\frac{1}{2}} + \frac{1}{12}F_{y2}^{n+\frac{1}{2}} + \frac{1}{12}F_{y3}^{n+\frac{1}{2}} + \frac{1}{12}F_{y4}^{n+\frac{1}{2}} \\ \frac{1}{6}F_{y1}^{n+\frac{1}{2}} + \frac{1}{12}F_{y2}^{n+\frac{1}{2}} + \frac{1}{12}F_{y3}^{n+\frac{1}{2}} + \frac{1}{12}F_{y4}^{n+\frac{1}{2}} \\ \frac{1}{6}F_{y1}^{n+\frac{1}{2}} + \frac{1}{12}F_{y2}^{n+\frac{1}{2}} + \frac{1}{12}F_{y3}^{n+\frac{1}{2}} + \frac{1}{12}F_{y4}^{n+\frac{1}{2}} \end{bmatrix}$$

$$
-\Delta t n_z A \begin{bmatrix} \frac{1}{6}F_{z1}^{n+\frac{1}{2}} + \frac{1}{12}F_{z2}^{n+\frac{1}{2}} + \frac{1}{12}F_{z3}^{n+\frac{1}{2}} + \frac{1}{12}F_{z4}^{n+\frac{1}{2}} \\ \frac{1}{6}F_{z1}^{n+\frac{1}{2}} + \frac{1}{12}F_{z2}^{n+\frac{1}{2}} + \frac{1}{12}F_{z3}^{n+\frac{1}{2}} + \frac{1}{12}F_{z4}^{n+\frac{1}{2}} \\ \frac{1}{6}F_{z1}^{n+\frac{1}{2}} + \frac{1}{12}F_{z2}^{n+\frac{1}{2}} + \frac{1}{12}F_{z3}^{n+\frac{1}{2}} + \frac{1}{12}F_{z4}^{n+\frac{1}{2}} \\ \frac{1}{6}F_{z1}^{n+\frac{1}{2}} + \frac{1}{12}F_{z2}^{n+\frac{1}{2}} + \frac{1}{12}F_{z3}^{n+\frac{1}{2}} + \frac{1}{12}F_{z4}^{n+\frac{1}{2}} \end{bmatrix}
$$

The consistent mass matrix is :

$$
M_C = \int_\Omega N_j N_k d\Omega = \int_\Omega \begin{bmatrix} N_1N_1 & N_1N_2 & N_1N_3 & N_1N_4 \\ N_2N_1 & N_2N_2 & N_2N_3 & N_2N_4 \\ N_3N_1 & N_3N_2 & N_3N_3 & N_3N_4 \\ N_4N_1 & N_4N_2 & N_4N_3 & N_4N_4 \end{bmatrix} d\Omega = \frac{V}{20}\begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}
$$

Summing up rows and puutting the result on the diagonal gives the lumped mass matrix :

$$
M_L = \frac{V}{20}\begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix}
$$

A7 2D Tangents and Normals

In order to evaluate the boundary integrals and also the normal and tangential velocities it is necessary to calculate the normal (**n**) and tangent (**t**) to an edge of an element.

The edge for which it is necessary for the normal to be calculated is r_{ij} directed in an anticlockwise

direction around the triangular element. This lies in the xy plane. The unit outward normal can then be determined by taking the cross product of r_{ij} with the z axis (k) and dividing by its magnitude.

$$n = \frac{r_{ij} \times k}{|r_{ij} \times k|}$$

The tangent vector can be evaluated as follows :

$$t = \frac{k \times n}{|k \times n|} = \frac{r_{ij}}{|r_{ij}|}$$

The normal and tangential velocities can be calculated from the normal and tangential vectors as follows

$$\begin{bmatrix} v_n \\ v_t \end{bmatrix} = \begin{bmatrix} n_x & n_y \\ t_x & t_y \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

A8 3D tangents and Normals

The normal to any triangular planar surface is easily obtained by calculating the cross product of any two sides of the triangle. It is now necessary to determine two tangential vectors to that normal. There are an infinite number of tangent vectors to the surface. The first tangent vector is obtained from calculating the cross product of the normal vector with either the k or the j axis.

$$n \times k = \begin{vmatrix} i & j & k \\ n_x & n_y & n_z \\ 0 & 0 & 1 \end{vmatrix}$$

$$n \times j = \begin{vmatrix} i & j & k \\ n_x & n_y & n_z \\ 0 & 1 & 0 \end{vmatrix}$$

if n and k are not parallel, that is

$$|n \times k| > 0.001$$

the k axis is used as reference. The first tangent vector is then calculated as follows :

$$t1 = \frac{n \times k}{|n \times k|}$$

otherwise

$$t1 = \frac{n \times j}{|n \times j|}$$

The second tangent vector can then be calculated as follows

$$t2 = n \times t1$$

A9 Error Indicator

The error indicator from section 4.2 is :

$$E_i = \sqrt{\frac{\sum_{k,l} \left(\int_\Omega \frac{\partial N_i}{\partial x_k}\frac{\partial N_j}{\partial x_l} d\Omega U_j\right)^2}{\sum_{k,l} \left(\int_\Omega \left|\frac{\partial N_i}{\partial x_k}\right| \left[\left|\frac{\partial N_i}{\partial x_l} U_j\right| + \epsilon\left(\left|\frac{\partial N_i}{\partial x_l}\right| |U_j|\right)\right] d\Omega\right)^2}}$$

The evaluation of the different terms in this error indicator is as follows :

$$\int_{\Omega} \frac{\partial N_I}{\partial x_k}\frac{\partial N_J}{\partial x_l} d\Omega\, U_J = A \begin{bmatrix} \frac{\partial N_1}{\partial x_k}\frac{\partial N_1}{\partial x_l} & \frac{\partial N_1}{\partial x_k}\frac{\partial N_2}{\partial x_l} & \frac{\partial N_1}{\partial x_k}\frac{\partial N_3}{\partial x_l} \\ \frac{\partial N_2}{\partial x_k}\frac{\partial N_1}{\partial x_l} & \frac{\partial N_2}{\partial x_k}\frac{\partial N_2}{\partial x_l} & \frac{\partial N_2}{\partial x_k}\frac{\partial N_3}{\partial x_l} \\ \frac{\partial N_3}{\partial x_k}\frac{\partial N_1}{\partial x_l} & \frac{\partial N_3}{\partial x_k}\frac{\partial N_2}{\partial x_l} & \frac{\partial N_3}{\partial x_k}\frac{\partial N_3}{\partial x_l} \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \end{bmatrix}$$

$$\int_{\Omega} \left| \frac{\partial N_I}{\partial x_k} \right| \left| \frac{\partial N_J}{\partial x_l} U_J \right| d\Omega = A \begin{bmatrix} \left| \frac{\partial N_1}{\partial x_k} \right| \left| \frac{\partial N_1}{\partial x_l} U_1 + \frac{\partial N_2}{\partial x_l} U_2 + \frac{\partial N_3}{\partial x_l} U_3 \right| \\ \left| \frac{\partial N_2}{\partial x_k} \right| \left| \frac{\partial N_1}{\partial x_l} U_1 + \frac{\partial N_2}{\partial x_l} U_2 + \frac{\partial N_3}{\partial x_l} U_3 \right| \\ \left| \frac{\partial N_3}{\partial x_k} \right| \left| \frac{\partial N_1}{\partial x_l} U_1 + \frac{\partial N_2}{\partial x_l} U_2 + \frac{\partial N_3}{\partial x_l} U_3 \right| \end{bmatrix}$$

$$\int_{\Omega} \left| \frac{\partial N_I}{\partial x_k} \right| \left| \frac{\partial N_J}{\partial x_l} \right| |U_J| d\Omega = A \begin{bmatrix} \left| \frac{\partial N_1}{\partial x_k} \right| \left(\left| \frac{\partial N_1}{\partial x_l} \right| |U_1| + \left| \frac{\partial N_2}{\partial x_l} \right| |U_2| + \left| \frac{\partial N_3}{\partial x_l} \right| |U_3| \right) \\ \left| \frac{\partial N_2}{\partial x_k} \right| \left(\left| \frac{\partial N_1}{\partial x_l} \right| |U_1| + \left| \frac{\partial N_2}{\partial x_l} \right| |U_2| + \left| \frac{\partial N_3}{\partial x_l} \right| |U_3| \right) \\ \left| \frac{\partial N_3}{\partial x_k} \right| \left(\left| \frac{\partial N_1}{\partial x_l} \right| |U_1| + \left| \frac{\partial N_2}{\partial x_l} \right| |U_2| + \left| \frac{\partial N_3}{\partial x_l} \right| |U_3| \right) \end{bmatrix}$$

APPENDIX B

B.1 Introduction

The usefulness of a numerical code not only depends on its solution ability, but also on the infrastructure within which it operates. This typically involves pre and post-processing routines which include mesh generation and the ability to view the solution data in some form. In this appendix, the contour plotting algorithms, the 3D mesh generation algorithms and three dimensional viewing algorithms used in this research will be described. The 3D viewing algorithms described here are extensions of techniques described in Angell and Griffith[44].

B.2 Two Dimensional Contour Plotting

After every timestep of the solver a new set of solutions is obtained on the finite element grid. The solution is solved at the nodal points with the solutions within each element being obtained from linear finite element interpolations (see appendix A). The contour plotting routine will plot either constant density (isopycnic) or constant pressure (isobaric) lines in the flow domain.

The contour interval is specified by the user. Each contour line representing a constant value ϕ is drawn as follows :

Loop through each element

If a contour crosses an element, it will cut two of the three edges of the element. Each edge with endpoints i and j, needs to be tested to determine whether it is crossed by the contour line. If the nodes defining each edge have values ϕ_i and ϕ_j, the contour will cross that edge if $\phi < \max(\phi_i, \phi_j)$ and $\phi > \min(\phi_i, \phi_j)$. The coordinates of the intersection of the contour and edge is obtained by simple linear interpolation.

$$x = \frac{(x_j - x_i)(\phi - \phi_i)}{(\phi_j - \phi_i)} + x_i, \qquad y = \frac{(y_j - y_i)(\phi - \phi_i)}{(\phi_j - \phi_i)} + y_i$$

The two intersection points on the two edges can then be joined by a straight line.

B3 Three Dimensional Mesh Generation

The parametric mesh generation scheme described in Zienkiewicz and Philips[29] and Zienkiewicz[30] is used to generate the finite element meshes. The geometry is first subdivided by the user into hexagonal blocks, each with its own parametric coordinate system. (fig B.1)

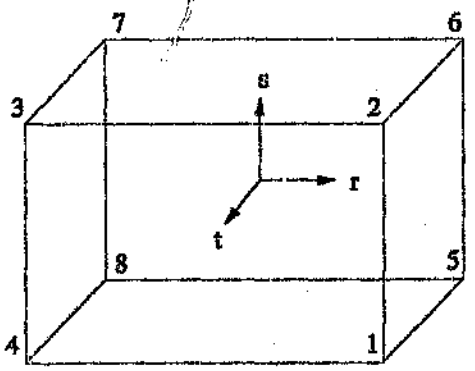


Figure B.1 Hexagonal parametric block

Mappings between parametric and real coordinates are as follows :

$$x = N_i x_i$$
$$y = N_i y_i$$
$$z = N_i z_i$$

where

$$N_1 = \frac{1}{8}(1 + r)(1 - s)(1 + t) , \quad N_2 = \frac{1}{8}(1 + r)(1 + s)(1 + t)$$
$$N_3 = \frac{1}{8}(1 - r)(1 + s)(1 + t) , \quad N_4 = \frac{1}{8}(1 - r)(1 - s)(1 + t)$$
$$N_5 = \frac{1}{8}(1 + r)(1 - s)(1 - t) , \quad N_6 = \frac{1}{8}(1 + r)(1 + s)(1 - t)$$
$$N_7 = \frac{1}{8}(1 - r)(1 + s)(1 - t) , \quad N_8 = \frac{1}{8}(1 - r)(1 - s)(1 - t)$$

The user specifies the number of elements to be generated in each parametric coordinate direction (nr, ns, nt) respectively. The element length in each direction in parametric space is calculated as follows:

$$dr = \frac{2}{nr}$$
$$ds = \frac{2}{ns}$$
$$dt = \frac{2}{nt}$$

The block is subdivided into smaller hexagonal blocks as follows.

```
r1 = s1 = t1 = -2.0
Loop : r2 = r1 + dr
  Loop : s2 = s1 + ds
     Loop : t2 = t1 + ds
        At this point a block is fully defined (r1, r2, s1, s2, t1, t2). The coordinates in the cartesian
        coordinate system can be calculated using the parametric mapping equations defined above.
        t1 = t2
     Next t
      t1 = -2.0
     s1 = s2
  Next s
  s1 = -2.0
  r1 = r2
Next r
```

Each block can now be divided into tetrahedra as in fig B.2.

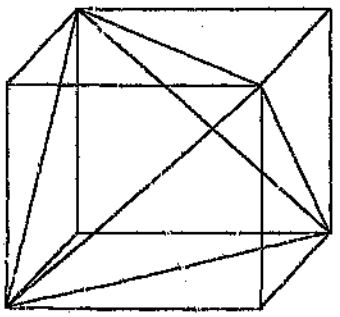


Figure B.2 Subdivision of a brick element into five tetrahedra

B4 Three Dimensional Viewing Algorithms

The scene to be viewed consists of triangular facets in three dimensional space. Appropriate translations and rotations must be performed on the scene so that the scene can be displayed from the desired viewpoint. The scene must be then displayed with all hidden lines removed.

Data Structures :

An explicit edge list is used to store the facet data. Edges facet references an edge. Each edge then references two coordinates which make up the endpoints of the edge.

COORDS(1..NCOORDS, 1..3)	{The coordinates for each point}
FACETS(1..NFAC, 1..3)	{The three edges defining each facet}
EDGES(1..NEDGES, 1..2)	{The two coordinates defining each edge}

B 4.1 Coordinate Transformations

There are three coordinate systems involved in this algorithm. The first is the absolute coordinate system. The geometry to be viewed is described in this coordinate system. The observer system has its origin at the observers viewpoint (ex, ey, ez) with the negative z axis lined up with the viewing direction (dx, dy, dz). All the absolute coordinates are converted to the observer coordinate system. The observer coordinates are then projected either orthographically or perspectively onto a viewing plane.

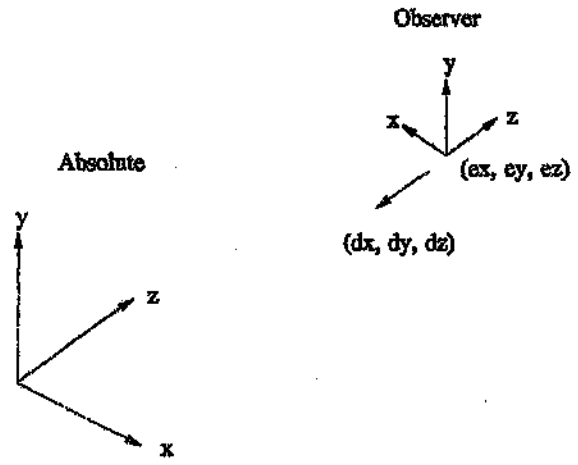


Figure B.3 Absolute and observer coordinates

The origin of the absolute coordinate system is first translated to (ex, ey, ez).

$$x' = x - ex$$
$$y' = y - ey$$
$$z' = z - ez$$

A series of rotations are performed on all the coordinates in order to line up the x, y and z axes as follows:

$$p' = Rp$$

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}, \quad R_y = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}, \quad R_z = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The axes are rotated about the z axis by angle $\alpha = \arctan(-dy/-dx)$.

The axes are the rotated about the y axes by angle $\beta = \arctan(dx^2+dy^2/-dz)$;

The axes are then rotated about the z axis by angle $\gamma = -\arctan(-dy.dz/dx^2+dy^2+dz^2)$

B 4.2 Projection

The coordinates can then be projected onto a viewing plane either orthographically or perspectively.

Orthographic projection :

$$x_p = x_o$$
$$y_p = y_o$$

Perspective projection :

$$x_p = -\frac{x_o d}{z_o}$$
$$y_p = -\frac{y_o d}{z_o}$$

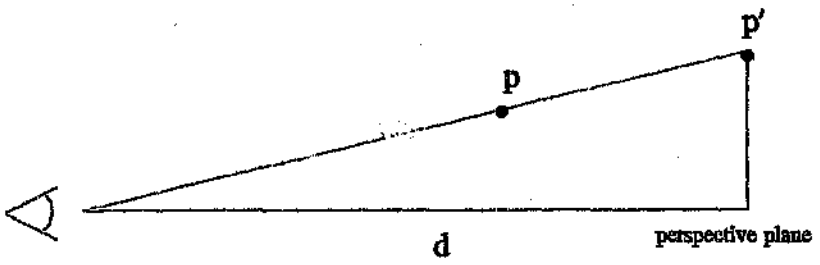


Figure B.4 Perspective projection of a point

B 4.3 Hidden line elimination

Before each edge in the scene can be drawn, its visibility must be determined. Edges may be completely visible, completely hidden, or only partially obscured.

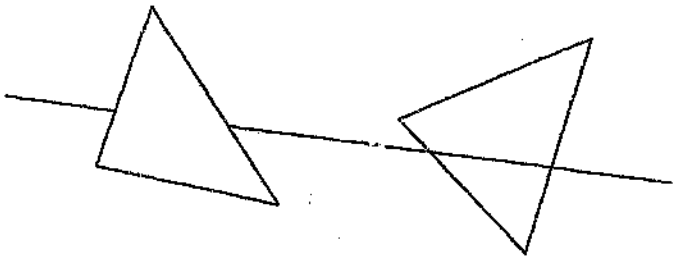


Figure B.5 A line partially obscured by a facet

Loop through facets

Check whether a facet must be displayed or not.

If a facet is not to be displayed than the edges are marked.

Loop through edges

Compare edge against every single facet in the scene. The whole edge may be visible, or the whole edge may be obscured. Parts of the edge may be visible and parts invisible. Only those parts of the edge that are visible are drawn.

These are essentially two dimensional calculations.

Whether an edge is obscured by a facet can be determined as follows :

The plane containing the facet can be described by the following vector equation :

$$n.a = k$$

For any point p on the plane.

$$f(p) = n_x x + n_y y + n_z z - k = 0$$

f(p) = 0 for any point on the plane but non zero for all other points. This plane divides space into a positive and negative set. If f(p1) and f(p2) both have the same sign, they will be on the same side of the plane. This way it can be determined whether a point on the line is on the same side of the plane as the eye (EX, EY, EZ). If the point on the line is on the opposite side of the plane then the section of the line is invisible and must not be drawn.

B5 Three Dimensional Contour Plotting

Three dimensional data is viewed by displaying the solution contours on a plane which slices the geometry.

The cutting plane defined by the user is expressed as follows

$$n.a = k$$

where n is a normal to the plane, a is a point on the plane and k is a constant.

Each tetrahedron in the mesh consists of six edges. All edges are tested to determine whether they

are intersected by the plane.

Any point on the edge ij can be defined in vector form as follows:

$$p = b + \mu d$$

where

$$b = (x_i , y_i , z_i)$$
$$d = (x_j - x_i , y_j - y_i , z_j - z_i)$$

The order of i and j does not matter.

$$n.(b + \mu d) = k$$

Solving for μ

$$\mu = \frac{k - n.b}{n.d}$$

If n.d = 0, the line and plane are parallel and no unique intersection can exist.
If the line and plane are not parallel, they will intersect. However it is only the line segment between i and j that is of interest, not the entire line. The plane and line segment intersect if $0 <= \mu <= 1$.

The plane will either intersect at three or four points, forming a planar triangle or quadrilateral. If a quadrilateral is created, it is divided into two triangles. Once all the edges have been tested a set of planar triangles will exist. The values at each node will have been calculated. The contour plotting proceeds as in the two dimensional case.

REFERENCES

1. M. Sommerfeld and H. M. Müller, 'Experimental and numerical studies of shock wave focussing in water', *Experiments in Fluids*, 6, 209-216 (1988).

2. H. M. Müller, 'Stosswellenfokussiering in wasser', Phd Thesis, University of Aachen, (1987).

3. R. W. MacCormack, "The effect of viscosity in hypervelocity impact cratering.", Tech. Paper AIAA-69-354 (1969).

4. R. M. Beam and R. F. Warming, "An implicit factored scheme for the compressible Navier-Stokes equations", AIAA j., 16, 393-402 (1978).

5. C. Hirsch, Numerical Computation of Internal and External Flows, Volume 1, John Wiley and Sons (1988).

6. C. Canuto, M. Y. Hussaini, A. Quarteroni and T. A. Zang, Spectral Methods in Fluid Dynamics, Springer-Verlag (1988).

7. Wai Sun Don, 'Numerical study of pseudospectral methods in shock wave applications.', *J. Comp. Physics*, 110, 103-111 (1994).

8. A. J. Baker, Finite Element Computational Fluid Dynamics, McGraw-Hill Book Company (1983).

9. J. Quirk, 'An alternative to unstructured grids for computing gas dynamic flows around arbitrarily complex two dimensional bodies.', *Computers and Fluids*, 23, 125-140 (1994).

10. W. E. Dietz, J. L. Jacobs and J. L. Fox, 'Application of domain decomposition to the analysis of complex aerodynamic configurations', *Proceedings of the Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, 428 - 450. (1990).

11. K. Z. Korczak and A. T. Patera, "An isoparametric spectral element method for solution of the

Navier-Stokes equations in complex geometry", *J. Comp. Pysics, 62, 361-382 (1986).*

12. J. A. Benek, P. G. Buning and J. L. Steger, 'A 3-D chimera grid embedding technique', *AIAA 83-1944*, (1993).

13. T. J. Kirsten, 'Time accurate shock wave calculation using an implicit approximate factorisation in generalised curvilinear coordinates' in Proceedings of the third national symposium on Computational Fluid Dynamics, presented by the Unit for Computational Mechanics and Bureau for Continuing Education, Potchefstroom University for CHE, June 1993.

14. C. Bruneau, 'Computation of hypersonic flows by a finite element least squares method', in *Proc. 7th Int. Conference on Finite Elements in Fluids*, University of Alabama Press. Huntsville, 757-762, (1989).

15. M. Mallet, 'A finite element method for computational fluid dynamics', Phd thesis, Stanford University (1986).

16. F. P. Brueckner and J. C. Heinrich, "Petrov-Galerkin finite element model for compressible flows", *Int. j. numer. methods eng.*, 32, 255-274 (1991).

17. J. Donea, "A Taylor-Galerkin method for convective transport problems", *Int. j. numer. methods eng.*, 20, 101-119 (1984).

18. R. Löhner, K. Morgan and O. C. Zienkiewicz, "The solution of non-linear hyperbolic equation systems by the finite element method", *Int. j. numer. methods fluids*, 4, 1043-1063 (1984).

19. R. Löhner, K. Morgan and O. C. Zienkiewicz, "An adaptive finite element procedure for compressible high speed flows", *Comput. meths. appl. mech. eng.* 51, 441-465 (1985).

20. J. Peraire, J. Peiro, L. Formaggia, K. Morgan and O. C. Zienkiewiecz, "Finite element Euler computations in three dimensions", *Int. j. numer. methods eng.*, 26, 2135-2159 (1988).

21. O. Hassan, K. Morgan and J. Peraire, "An implicit finite element method for high-speed flows",

Int. j. numer. methods eng., 32, 183-205 (1991).

22. J. P. Boris and D. L. Book, 'Flux-corrected transport. I. SHASTA, a transport algorithm that works', *J. Comp. Phys.*, 11, 36-69, (1973).

23. D. L. Book, J. P. Boris and K. Hain, 'Flux-corrected transport. II. Generalisations of the method', *J. Comp. Phys.*, 18, 248-283 (1975).

24. J. P. Boris and D. L. Book, 'Flux-corrected transport. III. Minimal-error FCT algorithms', *J. Comp. Phys.*, 20, 397-341 (1976).

25. S. T. Zalesak, 'Fully multidimensional flux-corrected transport algorithm for fluids, *J. Comp. Phys.*31, 335-362 (1979).

26. G. Erlebacher, 'Solution adaptive triangular meshes with applications to the simulation of plasma equilibrium', Ph.D thesis, Columbia University (1984).

27. A. K. Parrot and M. A. Christie, 'FCT applied to the 2-D finite element solution of tracer transport by single phase flow in a porous medium', in K. W. Morton and M. J. Baines (eds), *Numerical Methods for Fluid Dynamics*, Oxford University Press 609-620 (1986).

28. R. Löhner, K. Morgan, J. Peraire and M. Vahdati, "Finite element flux-corrected transport (FEM-FCT) for the Euler and Navier-Stokes equations", *Int. j. numer. methods fluids*, 7, 1093-1109 (1987).

29. O. C. Zienkiewicz and D. V. Philips, 'An automatic mesh generation scheme for plane and curved surfaces by isoparametric coordinates', *Int. j. numer. methods eng.* 3, 519-528 (1971).

30. O. C. Zienkiewicz, 'The Finite Element Method, third edition', McGraw-Hill Book Company (UK) Limited. (1977).

31. C. V. Ramakrishnan, S. Ramakrishnan, A. Kumar and M. Bhattacharya, 'An integrated approach for automatic generation of two/three dimensional finite element grids using spatial occupancy

enumeration and Delauney triangulation', *Int. j. numer. methodes eng.*, 34, 1035-1050 (1992).

32. S. H. Lo, "A new mesh generation scheme for arbitrary planar domains", *Int. j. numer. methods eng.* 21, 1403-1426 (1985).

33. J. Peraire, M. Vahdati, K. Morgan and O. C. Zienkiewiecz, "Adaptive remeshing for compressible flow calculations", *J. Comp. Phys.*, 98 , 449-466 (1987).

34. J. Probert, O Hassan, J. Peraire and K. Morgan, "An adaptive finite element method for transient compressible flows", *Int. j. numer. methods eng.*, 32, 1145-1159 (1991).

35. R. Löhner, "An adaptive finite element scheme for transient problems in CFD", *Comput. meths. appl. mech. eng.*, 61, 323-338 (1987).

36. R. Löhner, K. Morgan and O. C. Zienkiewicz, "Adaptive grid refinement for the compressible Euler equations", in Accuracy Estimates and Adaptive Refinements in Finite Element Computations, eds I. Babuška, O. C. Zienkiewicz and E. R. de A. Olivieira, John Wiley and Sons (1986).

37. F. L. Stasa, 'Applied Finite Element Analysis for Engineers', CBS Publishing, (1985).

38. W. I. Usab and E. M. Murman, "Embedded mesh solutions of the Euler equations using a multiple-grid method", in W. G. Habashi (ed), *Advances in Computational Transonics*, Pineridge Press, Swansea, p 447, (1985).

39. G. Ben-Dor and I.I. Glass, 'Domains and boundaries of non-stationary shock wave reflexions. 2. Monatomic Gas', *J. Fluid Mech.*, 96, 735-756, (1978).

40. Frank M. White, 'Fluid Mechanics 2nd ed', McGraw Hill, (1986).

41. J. E. A. John, 'Gas Dynamics', Allan and Bacon, (1969).

42. S. Sivier, J. Baum, E. Loth and R. Löhner, 'Vorticity produced by shock wave diffraction' in

Shock Waves, Proceedings of the 18th International Symposium on Shock Waves., 143 - 150, (1991).

43 K. Takayama and O. Inoue, 'Shock wave diffraction over a 90 degree sharp corner - Posters presented at 18th ISSW', *Shock Waves*, 1, 301 - 312, (1991).

44. Ian O. Angell and Gareth Griffith, 'High Resolution Computer Graphics using FORTRAN 77', MacMillan Education Limited, (1987).