INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 25, 81–104 (1997)

A 3D UNSTRUCTURED MESH ADAPTATION ALGORITHM FOR TIME-DEPENDENT SHOCK-DOMINATED PROBLEMS

W. SPEARES AND M. BERZINS*

School of Computer Studies and Centre for CFD, University of Leeds, Leeds LS2 9JT, U.K.

SUMMARY

In this paper we present a tetrahedron-based, h-refinement-type algorithm for the solution of problems in 3D gas dynamics using unstructured mesh adaptation. The mesh adaptation algorithm is coupled to a cell-centred, Riemann problem-based, finite volume scheme of the MUSCL type, employing an approximate Riemann solver. The adaptive scheme is then used to compute the diffraction of shock waves around a box section corner for subsonic and supersonic post-shock flow. In the subsonic case, preliminary measurements of vortex filament speed and vortical Mach number are in broad quantitative agreement with known theoretical results. © 1997 by John Wiley & Sons, Ltd.

Int. J. Numer. Meth. Fluids, **25**: 81–104 (1997).

No. of Figures: 20. No. of Tables: 0. No. of References: 30.

KEY WORDS: 3D tetrahedral adaptation; time-dependent; shock capturing; shock diffraction

1. INTRODUCTION

The numerical investigation of phenomena associated with shock wave propagation through the use of conservative shock-capturing, high-resolution, Riemann problem-based numerical methods for hyperbolic conservation laws has generated great interest within the fluid dynamics community over recent years.[1] A number of high-quality two-dimensional numerical simulations of inviscid flows have been performed (e.g. References 2 and 3), which in some cases have even indicated the presence of new regimes in the physics of shock behaviour.[4] In order to be able to examine shock-dominated processes at high spatial resolutions without incurring heavy computational penalties, even in two dimensions, it is desirable to use some form of mesh adaptation. Adaptive codes make use of the local flow field solution itself to determine where the high spatial mesh resolution is required and then employ some strategy to increase the grid resolution in those regions. This enables the high spatial mesh resolution to be concentrated around the important flow features (e.g. shocks, vortices and so on) rather than being wasted on parts of the computational domain where the flow activity is relatively unimportant. One increasingly popular approach to mesh adaptation is so-called *h-refinement*, where additional mesh nodes and elements are inserted into the computational domain in regions where the greater resolution is required and then removed from the mesh when the higher mesh node density is judged to be redundant. Examples of this approach on structured and unstructured meshes include References 5–7. It is important to extend this numerical work on shock behaviour to three dimensions to provide additional insight into the complex flow structures which shock propagation produces and, where possible, to make comparisons with available experimental

* Correspondence to: M. Berzins, School of Computer Studies and Centre for CFD, University of Leeds, Leeds LS2 9JT, U.K.

CCC 0271–2091/97/010081–24 $17.50 *Received November 1995*

and theoretical results. For good shock resolution in three dimensions the use of mesh adaptation methods is made almost mandatory by the prohibitive computational expense of specifying a uniformly fine mesh throughout the flow field.

In this paper we present a new h-refinement-type algorithm for 3D tetrahedron-based mesh adaptation, which in part extends ideas contained in References 8 and 9. One direct desirable feature of the approach is that the degradation in element quality due to the adaptation process remains bounded. Details of the construction of the adaptation algorithm are given along with performance figures and an outline of some useful extensions. The algorithm is coupled to a higher-order extension of the upwind Godunov scheme of the MUSCL type[10] on tetrahedra for the unsteady 3D Euler equations, employing an approximate Riemann solver of the HLLC type.[11] The adaptive solver is then used to make a preliminary investigation of 3D shock wave diffraction outwards from within a region with cuboid cross-section through a 2π solid angle. Two shock Mach numbers are examined. The first, Mach 3, causes supersonic post-shock flow, which results in the formation of a recompression shock as observed in 2D calculations.[3] The second, Mach 1·7, giving subsonic post-shock flow, shows the the formation of a clear ring vortex structure around the outer edge of the entrance cavity. The behaviour of these compressible inviscid ring vortices is of significant theoretical interest[12] and preliminary quantitative measurements are made of the vortical Mach number and velocity.

The paper is organized as follows. In Section 2 we state the computational problem and give a description of the numerical method employed. Section 3 gives the adaption algorithm, along with some constructional and performance-related details. In Section 4 we present numerical solutions. Finally we make some concluding remarks and indicate planned future work.

2. TIME-DEPENDENT GAS DYNAMICS

2.1. Equations

The gas dynamical shock diffraction problem that we wish to study here can in the first instance be adequately modelled by the inviscid 3D Euler equations. These fall into the general class of hyperbolic conservation laws of the form

$$U_t + [F(U)]_x + [G(U)]_y + [H(U)]_z = 0 \tag{1}$$

for three space dimensions (x, y, z) and with time t. The variable $U(x, y, z, t)$ is the vector of conserved variables and the vector functions $F(U)$, $G(U)$ and $H(U)$ are the analytic fluxes. On account of the need to admit discontinuous solutions such as shock waves and contact surfaces, it should be understood that we investigate weak solutions of the integral form of these equations,

$$\frac{\partial}{\partial t}\int\int\int_V U \,\mathrm{d}\tau + \int\int_{\partial V} (F\mathbf{i} + G\mathbf{j} + H\mathbf{k}) \cdot \mathrm{d}\mathbf{S} = 0, \tag{2}$$

where V represents some fixed control volume with volume element $\mathrm{d}\tau$ and surface ∂V with directed area element $\mathrm{d}\mathbf{S}$ and where $\mathbf{i}$, $\mathbf{j}$ and $\mathbf{k}$ are the Cartesian unit base vectors. For the case of the 3D Euler equations the conserved variables and fluxes may be written as

$$U = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E \end{bmatrix}, \qquad F = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ u(E+p) \end{bmatrix}, \qquad G = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ v(E+p) \end{bmatrix}, \qquad H = \begin{bmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ w(E+p) \end{bmatrix}, \tag{3}$$

where ρ is the density, u, v and w are the Cartesian x, y and z velocity components respectively and E is the total energy. This is defined as

$$E = \tfrac{1}{2}\rho(u^2 + v^2 + w^2) + e\rho, \tag{4}$$

where e is the specific internal energy assuming an ideal gas equation of state,

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

and which closes the system.

The initial conditions are set by specifying Rankine–Hugoniot shock data $\mathbf{U}_{\text{shock}}$ to the left of a box section surface of discontinuity at $t = 0$ and an ambient stationary state $\mathbf{U}_0$ elsewhere in the domain. The details of this are given in Section 4. Below we now describe the numerical scheme.

2.2. Numerical scheme

The numerical method we employ is a second-order-accurate, conservative cell-centred finite volume extension of Godunov's Riemann problem-based scheme,[13] using MUSCL-type piecewise linear reconstructions of the primitive variables within each mesh element.[10] The numerical solution in some element i at time t^n is denoted by U_i^n and is understood to be an approximation to the exact element-averaged volume integral of the solution, i.e.

$$U_i^n \approx \frac{1}{V_i}\int\int\int_{V_i} U(x, y, z, t^n)\mathrm{d}\tau, \tag{6}$$

where V_i is the volume of element i and is usually regarded as being valued at the element centroid for cell-centred schemes. Application of the integral conservation law (2) shows that the numerical solution at the next time level t^{n+1} may be written as

$$U_i^{n+1} = U_i^n - \frac{\Delta t}{V_k}\sum_{k=0}^{3} A_k \mathbf{F}_k \cdot \mathbf{n}_k, \tag{7}$$

where the sum is over the k faces of element i. The $\mathbf{n}_k$ are the outward face unit normal vectors and the A_k are the face areas. The fluxes $\mathbf{F}_k$ represent the numerical flux function for each element face, termed the element face fluxes, and are determined by the scheme. For the numerical scheme to be consistent, we require that the discretized solution in the presence of uniform flow reduce to the value of the analytic flux $\mathbf{F}(U)$, which gives the following condition on the numerical fluxes:

$$\mathbf{F}_k = \mathbf{F}_k(U_j, \ldots, U_i, \ldots, U_l),$$

where j and l are positive integers and

$$\mathbf{F}_k(U, \ldots, U) = \mathbf{F}(U).$$

In the case of the well-known Godunov scheme[13] these element face numerical fluxes are constructed from the solution of the local element Riemann problem (RP) at each element face. The inter-element RP has local initial data

$$U(x, 0) = \begin{cases} U_{\text{l}} & \text{if} \quad x < 0, \\ U_{\text{r}} & \text{if} \quad x > 0, \end{cases} \tag{8}$$

where U_{l} and U_{r} represent the left and right element data values on either side of a particular face, where x is a local normal co-ordinate in the frame normal to the face, at time $t = 0$. We denote the

solution to this RP by $U^*(U_l, U_r; x/t)$. The solution is self-similar in the x/t plane, being constant along rays through the origin in the local co-ordinate system (Figure 1).

The rotational invariance of the Euler equations leads to the identity

$$\mathbf{F}_k \cdot \mathbf{n}_k = \mathbf{R}^{-1} H(\mathbf{R}U). \tag{9}$$

Here $\mathbf{R}(\theta, \phi, \psi) = (\mathbf{\Psi}) \cdot (\phi) \cdot \mathbf{\Theta}(\theta)$ is the rotation matrix constructed from the Euler angles θ, ϕ and ψ associated with the given element face normal vector, where

$$\mathbf{\Theta}(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 & 0 \\ 0 & -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

$$\mathbf{\Phi}(\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \cos\phi & \sin\phi & 0 \\ 0 & 0 & -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \tag{10}$$

$$\mathbf{\Psi}(\psi) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos\psi & \sin\psi & 0 & 0 \\ 0 & -\sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

and H is the z-direction Euler flux. In fact, the $\mathbf{\Psi}$-matrix is just a rotation about the face normal vector and so effectively may be dispensed with. It is natural therefore to define the numerical flux, using the solution of the local inter-element RP in the frame normal to the face, as

$$\mathbf{F}_k \cdot \mathbf{n}_k = \mathbf{R}^{-1} H(\mathbf{R}U^*(U_l, U_r; 0)), \tag{11}$$

where now the RP solution is taken along the ray $x/t = 0$ (see Figure 1). If piecewise constant element-averaged data are used for the left and right data states U_l and U_r, then this numerical flux gives the first-order upwind Godunov scheme. To extend the scheme to second-order accuracy, one approach is to make use of piecewise linear data reconstructions within each element. Schemes of this type are generically known as MUSCL schemes,[10] where now the left and right data states U_l and U_r

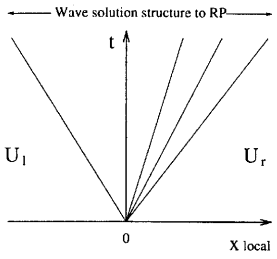

Figure 1. Local Riemann problem solution

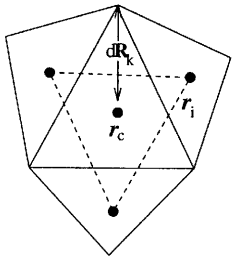

Figure 2. MUSCL scheme piecewise linear data reconstruction

represent face-interpolated variable values. The scheme is constructed in three stages. First a central difference gradient plane is fitted through the four surrounding tetrahedral neighbour elements for each primitive field variable (see Figure 2). That is, we find the plane for each primitive variable such that

$$U(\mathbf{r}), \quad \text{where} \quad U(\mathbf{r}_i) = U_i. \tag{12}$$

This gradient plane is then *limited* to be the maximum possible which does not produce undershoots or overshoots in the primitive field values when interpolated to the inter-element face centroid positions. Here we use the simplest possible strategy for doing this, extending to 3D the scalar central difference limiter described in Reference 14. The limiting amounts to finding a scalar α such that

$$\text{Min}\{U_i\} \leqslant U_c + \alpha \mathrm{d}\mathbf{R}_k \cdot \nabla U \leqslant \max\{U_i\}, \tag{13}$$

where $0 \leqslant k \leqslant 3$ and $\mathrm{d}\mathbf{R}_k$ is the displacement vector from the element centroid to the element vertices. This displacement vector is used so as to be sure to obtain the most extreme value of each field reconstruction along each element face.

The limited gradient plane, denoted $\nabla_\alpha U$, is then used to compute the face centroid-interpolated values of each primitive field within each element, which we denote by $\tilde{U}_{ik}$. That is,

$$\tilde{U}_{ik}^n = U_i^n + \mathrm{d}\mathbf{r}_k \cdot \nabla_\alpha U, \tag{14}$$

where now $\mathrm{d}\mathbf{r}_k$ is the displacement vector from the element centroid to the centroid of face k. Having obtained these face-interpolated values within each element, the second step in the scheme's construction is to perform a non-conservative predictor-type update—the 'Hancock step'[10]—on the data values within each element up to the half-time step level. That is, for each element i we now compute

$$U_i^{n+1/2} = U_i^n - \frac{\Delta t/2}{V_k} \sum_{k=0}^{3} A_k \tilde{\mathbf{F}}_k \cdot \mathbf{n}_k, \tag{15}$$

where the sum is again over the k faces of element i. The $\tilde{\mathbf{F}}_k$ represent here the analytic flux $F\mathbf{i} + G\mathbf{j} + H\mathbf{k}$ evaluated on the element i face k-interpolated field values $\tilde{U}_{ik}^n$. We note that this partial update is non-conservative, because the face-interpolated values used are local to each element and will in general change across each face.

The final stage in the construction of the numerical flux function is to use the partially updated element field values $U_i^{n+1/2}$ to re-evaluate the face-interpolated values using the *same* limited solution gradient $\nabla_\alpha U$. That is, we now compute new face-interpolated values

$$\tilde{U}_{ik}^{n+1/2} = U_i^{n+1/2} + \mathrm{d}\mathbf{r}_k \cdot \nabla_\alpha U. \tag{16}$$

These new face interpolants are then used as initial data to inter-element RPs across each face in the mesh. That is, for any two adjacent elements i and j which share a face k, we set $U_\mathrm{l} = \tilde{U}_{ik}^{n+1/2}$ and $U_\mathrm{r} = \tilde{U}_{jk}^{n+1/2}$, then compute the numerical flux for face k using (11). The solution update for each element is then computed using (7) as described above.

Implicit in this numerical method is the need to solve the RP for the Euler equations at each element interface at each time step. Rather than solve these inter-element RPs exactly, which is computationally expensive, a computationally inexpensive approximate Riemann solver is employed. We use the HLLC approximate solver, which is an improved version of the HLL solver of Harten *et al.*[15] and is obtained by incorporating the contact surface and shear waves into the wave pattern.[11] The solver works by resolving Rankine–Hugoniot shock relations across each wave in the RP solution, on the assumption that the wave speeds are known. This then gives simple expressions for the interwave states as functions of the wave speeds.[11] The satisfactory performance of the solver depends on obtaining good estimates of the wave speeds. Here we use estimates obtained from a primitive variable linearization of the equations.[16] The HLLC solver has been extensively tested and shown to be robust, accurate and computationally inexpensive.[11,17]

2.3. Choice of a stable time step

The time step is chosen by using a CFL-like condition based on an estimate of the maximum wave speed and element geometry.

While it is desirable to use the wave information from the solution of the local inter-cell RP as a means to establish a reliable stable time step size, in practice this would involve solution or approximate solution of the inter-element RPs prior to the predictor step, which would be computationally expensive. In fact, it is found that a far simpler strategy is sufficient. This is to use the solution data directly to obtain an estimate of the speed of the fastest local inter-cell wave. We do this by defining the time step to be

$$\Delta t = C_{\mathrm{CFL}} \times \mathrm{Min}\left(\frac{l_i}{V_i(U_i)}\right), \tag{17}$$

where l_i is the minimum edge length of element i and $V_{ij}(U_i)$ is the data-dependent estimate of the wave speeds for the faces surrounding element i. The constant C_{CFL} is the CFL coefficient. An effective choice for V_i is found to be

$$V_i = a_i + \sqrt{(u_i^2 + v_i^2)},$$

where a_i is the sound speed in element i and u_i and v_i are the x and y velocities respectively. This wave speed estimate has some justification in terms of the maximum speed of the characteristics in the pseudo-one-dimensional RPs. As in general shock waves will be present, we choose a conservative value for the CFL coefficient of 0·75 to give a margin for error with the estimation of the wave speeds. While it is recognized that this procedure for defining the time step is problem-dependent, it has the advantage of being easy to implement and, for the types of problems we are considering in this paper which involve only relatively weak shocks, sufficiently robust to produce good solutions.

2.4. Boundary conditions

For the unsteady shock propagation problems we employ two basic boundary conditions: a *reflective* condition and a *transmissive* condition. The boundary conditions are applied by specifying a layer of *ghost cells* surrounding the computational mesh. This amounts to attaching to each boundary face a set of flow variables, which may then be primed with the relevant data states and

used to construct a numerical boundary flux. For the case of a transmissive condition the data in the face storage are set equal to the data in the mesh element adjacent to the face. This results in a zero-wave-strength RP along the boundary.

For the reflective boundary condition the density and pressure variables of the ghost cells are identified as in the transmissive case, but the momentum vectors in the face storage are set equal to the reflection of the momentum vectors in the adjacent mesh element, where the reflection is carried out in the plane of the boundary. That is,

$$\mathbf{U}_{\text{ghost}} = \mathbf{R}^{-1}(\theta, \phi)\mathbf{M}\mathbf{R}(\theta, \phi)\mathbf{U}_{\text{wall}},$$

where $\mathbf{M}$ is the 5×5 matrix with leading diagonal $(1, 1, 1, -1, 1)$ and all other entries zero.

This results in a boundary RP whose contact and shear waves are stationary and lie along the boundary surface. Since for the construction of the central interpolant it is necessary to also include some geometrical information, the face boundary data are taken as being valued at the reflection in the plane of the boundary of the centroid of the element adjacent to the wall.

3. MESH ADAPTATION ALGORITHM

3.1. General description

The mesh refinement algorithm for tetrahedral adaptivity (TETRAD) extends previous work on unstructured 2D triangular meshes.[18] The approach we take is hierarchical in nature[6,8,9,18] and is applicable to meshes constructed from tetrahedron-shaped elements. The basic mesh geometrical objects of *nodes*, *edges*, *faces* and *elements*, which together form the computational domain, naturally map onto the data objects within the adaptation algorithm data structure. These data objects contain all flow and connectivity information sufficient to adapt the mesh structure and flow solution, either by a local mesh *refinement* or by a local mesh *derefinement* process. The mesh adaptation strategy assumes that there exists a 'good quality' initial unstructured tetrahedral meshing of the computational domain, which is taken to be the invariant base mesh of the region. The refinement process then adds nodes to this base level mesh by edge, face and element subdivision, with each change in the computational mesh being tracked within the code data structure by the construction of a data hierarchy. The derefinement process is the inverse process to refinement, where nodes, faces, edges and elements are removed from the mesh by working back up through the local mesh refinement hierarchy, recovering with each coarsening of the mesh the previous local mesh structure prior to its refinement, until the base mesh level is again encountered. In this approach no further coarsening of the mesh is possible past the base mesh level.

The process of mesh adaptation described is invoked automatically in response to some dynamically evolving flow solution criteria, with the regions of mesh refinement corresponding to regions of significant flow activity where it is desirable to have increased spatial resolution. How these criteria are chosen has important consequences for the overall operation of the adaptive solver and will be discussed below. The complete adaptive solver may be thought of as consisting of three parts, namely (i) the mesh data structure, (ii) the adaptation algorithm and (iii) the flow integration algorithm, where diagrammatically these objects are organized as

$$\boxed{\text{adaptation}} \rightarrow \boxed{\text{data structure}} \leftarrow \boxed{\text{integration}}.$$

Thus the adaptation and integration operations can be thought of as two distinct processes that are applied to the central data structure. The former alters the local connectivity in response to local flow solution features and the latter advances the flow field solution parameters in time, with the application of adaptation and integration operations being interleaved together in a suitable way by a

driver module. Below we now give a full account of how the data structure and adaption operation algorithm are constructed.

3.2. *Mesh data structure*

The choice of what data structure to use for the adaptive algorithm is a difficult one, it being to some extent influenced by considerations other than those connected purely with the construction of the adaption algorithm. Examples of these might be machine resource constraints, the type of integration schemes likely to be employed and the ease of parallelization of the algorithm. The overhead associated with 3D computation makes it unattractive to support more than a single mesh connectivity framework, but this clearly must be sufficient to make the refinement and derefinement operations straightforward to implement. Following other h-refinement algorithms,[6,8,9] we define a set of data objects or types. These may be implemented in, say, 'C' directly in the form of *structures* and *pointers*, though clearly an analogous approach can be taken within the framework of a language such as Fortran77 as well. We use the following (see Figure 3).

1. *An element object.* For tetrahedral elements these objects consist of four *node* objects corresponding to the vertices of the tetrahedron and six *edge* objects corresponding to the sides of the tetrahedron. The element object also contains *parent* and *child* pointers (see below).
2. *A face object.* All the faces of a tetrahedral element are defined by three *node* objects, each one corresponding to a vertex of the face. The face object also contains a link (pointer) to a single *element* object, as only boundary faces are stored.

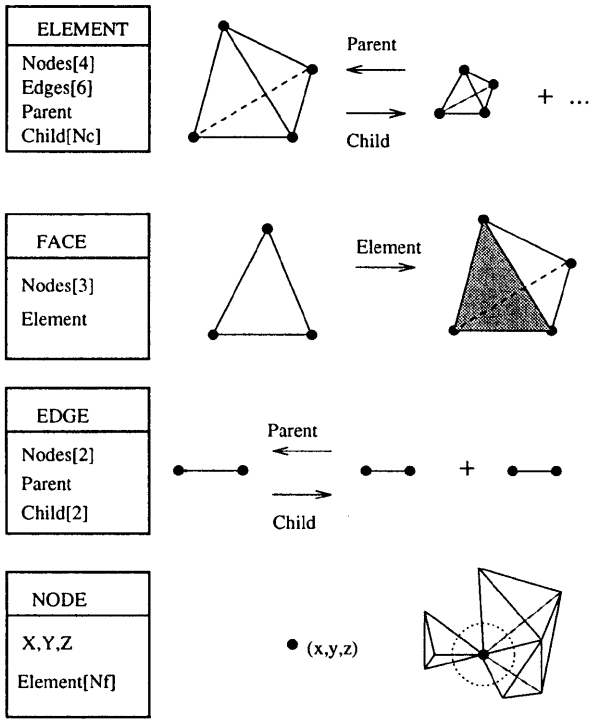

Figure 3. Mesh objects and data structure

3. *An edge object.* This is defined by two *node* objects and corresponds to an edge in the mesh, with edge *parent* and *child* pointers to support refinement (see below).
4. This corresponds to a vertex in the computational mesh and contains the co-ordinates of the vertex along with a link list of all the *element* objects corresponding to tetrahedral elements surrounding the vertex in the mesh.

The link list of all elements which contain a given node is termed the node's family and is stored in the form of a list of element pointers in each node object. This represents the fundamental connectivity supported by the data structure, from which all other mesh connectivities may be deduced. For example, all the elements which share a given edge may be determined by finding the common elements in the two families of elements surrounding the nodes which define the edge. Direct support of the node–element connectivity is a natural choice for use with cell vertex-based integration schemes, though other choices are possible; for example, see Reference 8, where the edge–element connectivity is explicitly supported.

The connectivity outlined above is sufficient to completely specify a given mesh, but it does not contain the connectivity required to construct the adaption hierarchy. For this some additional information is required, which in the case of elements and edges consists of storing *parent* and *child* addresses. The bisection of a parent edge by the addition of a node to the mesh results in the creation of two child edges. The situation is similar in the case of the elements, but there are now a number of possible child elements depending on how the element is dissected by the refinement process. This number must also be stored. The tree structure so defined is sufficient to completely specify the element refinement history for the purposes of adaption. Unfortunately, in 3D the same is not true of the edges. The action of element dissection in general will create new edges not derived by the subdivision of a previously existing parent but coming from the dissection of the element faces. We therefore use a set of link lists for the edge data structure, one for each level of refinement in the mesh. In this way, as extra edges are created at a given level, they may easily be incorporated into the data structure so as to completely specify the edge adaption hierarchy. Finally, the nodes and faces are also organized as link lists. For the nodes this is the natural choice of data structure. For the faces we use a link list rather than the more natural tree structure, as they are only required on the boundary of the mesh as a means of implementing the integration scheme. This is also the reason for the single element pointer within each face giving the address of the element with a face adjacent to the boundary of the domain. To extend the data structure to include numerical parameters such as the flow variables, we use a device from Reference 7, in which each fundamental data object, node, edge, etc., in addition to the connectivity described above, also contains a *void pointer*. This is storage for the address of a data object of *unspecified type* and may be used as a 'catch-all' for any additional data parameters required by the code. Thus, for a cell-centred scheme where the flow variables are associated with the mesh elements, the relevant parameters may be collected together and attached to the element void pointer. For a cell vertex scheme the natural point of attachment would be the node. Below we now describe the mesh adaptation algorithm.

3.3. Mesh adaption algorithm

The mesh adaption is driven by refining and derefining element edges.[8] Thus, if an edge is refined by the addition of a node along its length, then all the elements which share the (parent) edge under refinement must be refined. In the case of derefinement all the elements which share the node being removed must be derefined. Numerical criteria derived from the flow field will mark an edge to either refine, derefine or remain unchanged. It is necessary to make sure that the edges targeted for refinement or derefinement pass various conditions prior to their adaption.[8] These conditions effectively decouple the regions of mesh refinement from those of derefinement, meaning that, for

example, an element is not both derefined and then refined in the same adaption step. We return to this below.

For reasons of both tetrahedral quality control and algorithmic simplicity, only two types of element subdivision are allowed. This approach makes the adaption algorithm in some ways reminiscent of its structured counterparts (e.g. Reference 5). The first type of subdivision, which we call *regular subdivision*, is the popular subdivision by eight, with a new node bisecting each edge of the parent element. This amounts to removing the corners of the parent tetrahedron and then dissecting the central octahedron by four across a chosen interior diagonal. The choice of which interior diagonal to dissect is important, with generally the longest one being chosen,[6] although other approaches are possible[19] (see Figure 4(a)). The second type of dissection, *green subdivision*,[20] introduces an extra node into the parent tetrahedron, which is subsequently connected to all the parent vertices and any additional nodes which bisect the parent edges.[9] This provides an easy way of dealing with any pattern of parent edge refinement and gives a means of removing inconsistently connected or 'hanging' nodes without the introduction of additional edge refinement (see Figure 4(b)). The five refinement possibilities (if all the edges are refined, then the parent element is regularly refined) give rise to between six and 14 interior child green elements. These elements may be of poorer quality in terms of aspect ratio and dimensional measures than their regular counterparts, which leads us to impose the restriction that a *green element may not be further refined*. That is, if a green element has an edge targeted for refinement, then the green refinement of the parent element is replaced by a set of regularly refined child elements. Consequently, green elements always signal a change between mesh levels (except in the trivial case of a 'hole' in a refined level) and so act as an interface between changes in grid resolution. This ensures that the elements of poorer quality should never occur in regions of strong flow activity through the use of appropriate numerical adaption flagging criteria. The regular tetrahedral refinement analysis of Ong,[19] who shows that the degradation in mesh quality due to the refinement process remains bounded, is applicable to the algorithm. This is supported by numerical experiments in Section 3.4.

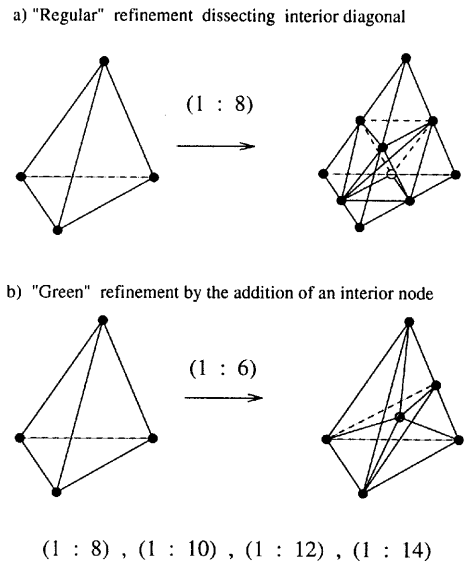

Figure 4. Mesh element refinement types

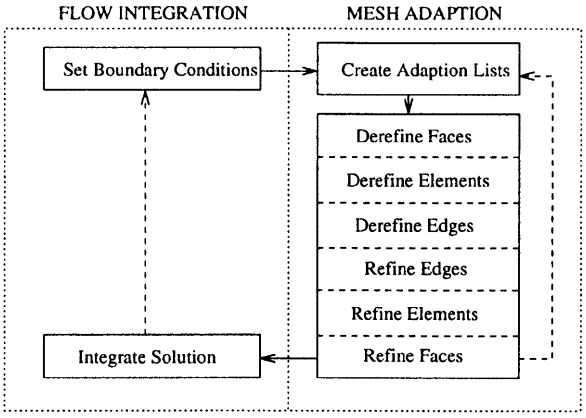

Figure 5. Mesh adaption sequence

The sequence of events involved in a complete adaption and integration of the mesh is shown in Figure 5. At first, mesh elements are flagged for adaption. This results in each mesh edge being targeted either for refinement, derefinement or no action. A set of lists of (pointers to) data objects which need to be adapted is then created. These lists are then passed to the appropriate function, processed and then passed to the next function. The motivation behind this approach is to make each step in the adaption sequence as transparent as possible.

To construct the list of edges and elements to be refined, each element adjacent to an edge targeted for refinement must be examined. If the element is green, then its parent element is placed on the list of elements whose child elements are to be derefined (since we exclude green refinement and therefore must replace the green elements by regular ones). This also involves placing any parent edges not already flagged for refinement on the edge refinement list, since for regular refinement all parent edges are refined. Furthermore, a search must be made for any green elements one level or more coarser in the mesh that lie adjacent to the green elements being replaced. To see the reason for this, consider Figure 6. The refinement of edge 'a' implies that the green tetrahedron T_1 must be processed. Since further refinement is ruled out, T_1 and its sibling tetrahedra must be replaced by a regular set. This in turn requires the refinement of edge 'b', with the same argument now applying to tetrahedron T_2 and so on. It is straightforward to code this process as a local recursive search, which is limited in extent by the local mesh refinement depth and not a cause of significant computational overhead.

Completion of this stage gives us the full set of edges and elements to be refined, along with the part of the element derefinement list associated with the green tetrahedra adjacent to refining edges. To obtain the rest of the edges to be derefined, some care is required. In order to construct the edge

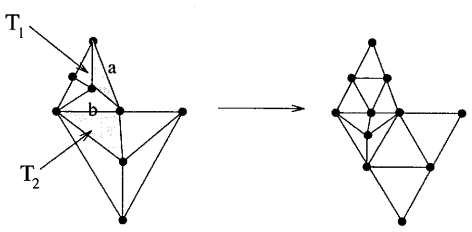

Figure 6. Refinement of adjacent green elements

and element derefinement list, a number of tests must be performed which effectively decouple the regions of refinement from those of derefinement, by restricting the number of edges targeted for derefinement. As in Biswas and Strawn's approach,[8] we impose the following restrictions on node removal.

1. Both child edges of the parent edge dissected by the node must be flagged to derefine.
2. No child element of the family of parent elements which connect to the node may itself be refined.
3. No edge in the connecting child element group may be flagged for refinement.
4. A node may not be removed if it is the vertex of any of the family of connecting parent elements.

The first three constraints bias the adaption algorithm towards refinement rather than derefinement, which is sensible on the grounds of solution accuracy. The fourth constraint takes care of the (unusual but possible) event of a node being completely surrounded by green refined parent elements and satisfying the first three derefinement constraints.

Following the above tests, the complete set of edges and elements to be derefined is available for processing. Faces on the domain boundary requiring derefinement are determined at the outset from the element derefinement list, while those requiring refinement are determined after the adaption of the mesh elements, edges and nodes is complete.

The difficulty of coding of the algorithm is greatly reduced by the use of a numbering convention for the nodes within each element, the edges within a refined parent element and so on. Fixed matrices may then be used as look-up tables to provide all the local connectivity information required for the reconstruction of the data structure during adaption. Below we now give some performance results for the algorithm.

3.4. Algorithm scaling behaviour

An important characteristic of any adaptive algorithm is its scaling behaviour in response to increases in the number of elements being processed (either refined or derefined). The ideal scaling behaviour would be a direct linear relationship between the number of elements being processed and the CPU time required. Below we give the results of two scaling tests performed on the algorithm.

Test 1: repeated refinement of one tetrahedral element. This is a basic test of the scaling behaviour of the refinement algorithm. A single tetrahedron is repeatedly refined in a *regular* manner, which results in from 8^0 to 8^7 tetrahedra being generated in 201·2 CPU seconds on a SGI 150 MHz R4400 processor. This gives a generation rate of approximately 10,500 tetrahedra each CPU second. Looked at graphically (Figure 7), plotting $\log_8$ of the CPU time against $\log_8$ of the number of tetrahedra, N_{tet}, shows that these timings scale as $8^{1\cdot04}$. Deviation from linear behaviour is shown in Figure 7 with 'test 1' representing the timing data versus the linear baseline 'linear 1'. The test demonstrates that the fundamental refinement process has close to $O(N)$ scaling behaviour and is not significantly affected by the mesh depth at which the refinement takes place. A more realistic scaling test of this is provided below.

Test 2: discontinuity refinement/derefinement test. This second test is more complex but also more typical of a realistic shock-dominated problem. Here a sheet linear discontinuity is propagated normally on a 3D test mesh and used to drive the adaption algorithm. This results in a cuboid region of mesh refinement which remains centred around the discontinuity at constant mesh depth, with the mesh derefining completely after the discontinuity has passed. This gives a complete test of the refinement and derefinement algorithm along with a realistic assessment of the algorithm's scaling

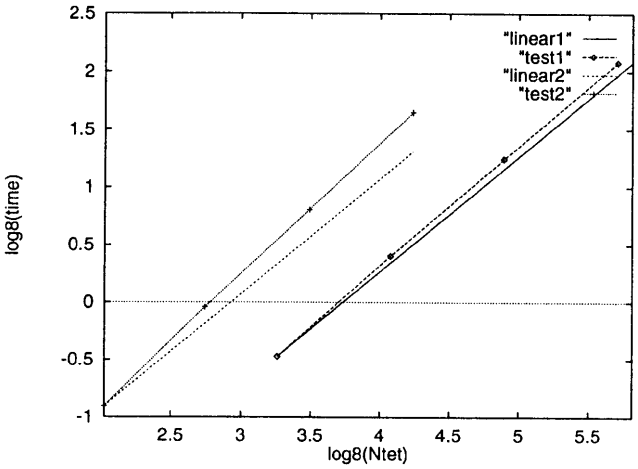

Figure 7. Algorithm scalability behaviour

behaviour. Again $\log_8(N_{\text{tet}})$ versus $\log_8(\text{time})$ is plotted, with maximum mesh depths of between one and four being employed around the discontinuity. The results are shown in Figure 7, with timings scaling as $8^{1{\cdot}14}$ for the unoptimized algorithm ('test 2' data as against the linear behaviour 'linear 2' in Figure 7). For a four-level 50,000-element mesh each complete refinement/derefinement cycle, involving the complete derefinement and then reconstruction of the mesh around the discontinuity, takes 11 CPU seconds. This compares consistently with the refinement timing results. The algorithm scaling has drifted slightly away from the optimal linear behaviour, but given that no attempt has been made yet to optimize the code, these results are quite encouraging.

3.5. Mesh quality behaviour

The issue of mesh quality degradation under the action of an adaptive algorithm is an important one and has been the subject of much recent work (e.g. References 19 and 21). A useful measure of mesh quality is the parameter α defined by[22]

$$\alpha = \frac{(\text{average element edge length})^3}{\text{element volume}}.$$

For a tetrahedron made up of edges of equal length, this quality measure gives a value of 8·48. For the tetrahedra which result from a five-way dissection of a cube with equal sides, we obtain a maximum of $\alpha = 10{\cdot}55$. Using a mesh constructed from this type of dissection, numerical experiments in the linear advection of a cuboid discontinuity verify that with one level of refinement the quality of the worst regular refined element has $\alpha = 14{\cdot}08$ and that of the worst green refined element has $\alpha = 53{\cdot}4$. Subsequent refinements cause no further degradation in the element quality as judged by this measure. Although the quality of the worst-case green tetrahedron is approximately 3·7 times worse than that of its poorest regular counterpart, such tetrahedra occur only at the interfaces between different levels in the mesh and so should not be a cause of significant solution degradation.

3.6. Adaption control mechanism

The Euler flow solver is combined with the adaptive algorithm by flagging regions of the mesh with (low) high density gradients for (de)refinement, with the calculation of local flow gradients

being performed across element faces. Where the face normal density gradient falls below or exceeds a chosen tolerance, the edges on the face are flagged to (de)refine. In addition, a 'safety layer' of refinement flagging is employed to ensure the full capture of solution discontinuities, which is the principal concern for this application. Likewise, a maximum mesh refinement depth is also specified. It would also be possible to use a local solution error indicator as a means of controlling the mesh adaptation (e.g. Reference 18). Coupling the adaption algorithm to the solution integrator is managed in a similarly straightforward manner. The CFL stability constraint on the integration time step (see below) ensures that shock waves cannot cross more than a fixed percentage (with the upper bound given by the CFL number) of a given computational cell in any one integration step. This, combined with the use of the flagging safety layer, means that a number of integrations may be performed before the shock is in danger of escaping the refined mesh region and adaption must occur. An estimate is provided by the minimum depth of refined cells ahead of the discontinuity (never less than two with the safety layer) divided by the CFL number (which we set at 0·75). Shock propagation experiments have indicated that in practice this estimate is quite conservative and that more steps may be taken in between mesh adaptions without significant degradation of the shock profile. Whenever mesh (de)refinement occurs, there is an issue of solution (de)construction within the new local mesh structure. Currently we use a simple piecewise constant averaging approach which ensures conservation, although a conservative limited piecewise linear approach similar to the structured case could be used (e.g. Reference 17).

3.7. Extension to include directional refinement

Before continuing, it is worth mentioning briefly two extensions to the adaption algorithm which may be made without great difficulty. The first is the incorporation of some form of directional refinement. Directional refinement is important to resolve well certain flow features such as boundary layers in viscous problems.[23] The morphology of the mesh adaptation scheme we have outlined above can be easily extended to include a directional refinement capability, simply by defining a second type of regular refinement based upon a directional dissection of the parent tetrahedra (see Figure 8).

Repeated application of this refinement pattern would result in the required aspect ratio increases in these special flow regions and would fit consistently with the green element removal of hanging nodes.

3.8. Support for hierarchical time refinement

The second extension to the adaption scheme we wish to outline is the support for the use of hierarchical integration methods. Here explicit use would be made of the mesh hierarchy as a means of obtaining computational gains within the flow integration. The essential observation is that the green elements signal a change between levels of refinement in the mesh and therefore may be used in a more general role as interface elements. This would make straightforward the application of, for example, hierarchical time refinement for explicit cell-centred schemes, similar to that carried out for

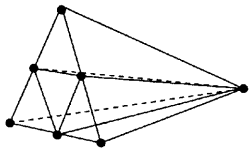

Figure 8. Regular directional refinement of elements

structures meshes,[5,7] except based around *cellular* rather than *patch* refinement. Specifically, the green elements could be used in the role of *dummy cells*,[17] not explicitly solved for by the integration method, but primed with flow data from an underlying coarse mesh solution via interpolation and acting as a boundary cell for the adjoining fine mesh level. This would make possible time-accurate advancement of the solution on a mesh-level basis, with each mesh leaf element being integrated with a locally determined time step size a number of times sufficient to keep it in step with its neighbouring elements. For structured hierarchical meshes this approach results in significant computational savings, but for the unstructured case the potential gains are less clear. Recently Batten *et al*.[24] reported computational gains approaching a factor of two using a two- dimensional bisection-based adaptation algorithm and hierarchical time refinement. While the approach outlined above has not yet been implemented, it is expected that making use of the green tetrahedral elements in this way will make the construction of the time refinement algorithm quite straightforward and without significantly higher overheads than the purely spatially adaptive code.

4. NUMERICAL RESULTS

The numerical simulations studied consist of shock wave diffraction around the 3D right-angled corner formed between two cuboid mesh regions, in the 3D analogue of the 2D case studied in Reference 3. The initial mesh is generated by hexagonal cell subdivision into 5184 tetrahedral elements, with the first cuboid domain having dimensions $0{\cdot}2 \times 0{\cdot}2 \times 0{\cdot}15$. Rankine–Hugoniot shock data initializes the computation, with the first cuboid domain being set to the post-shock data state and the discontinuity lying on the $x = 0{\cdot}2$ plane, at the interface between the two cuboid regions. The surface of the initial adapted mesh is shown in Figure 9, with the refined mesh region

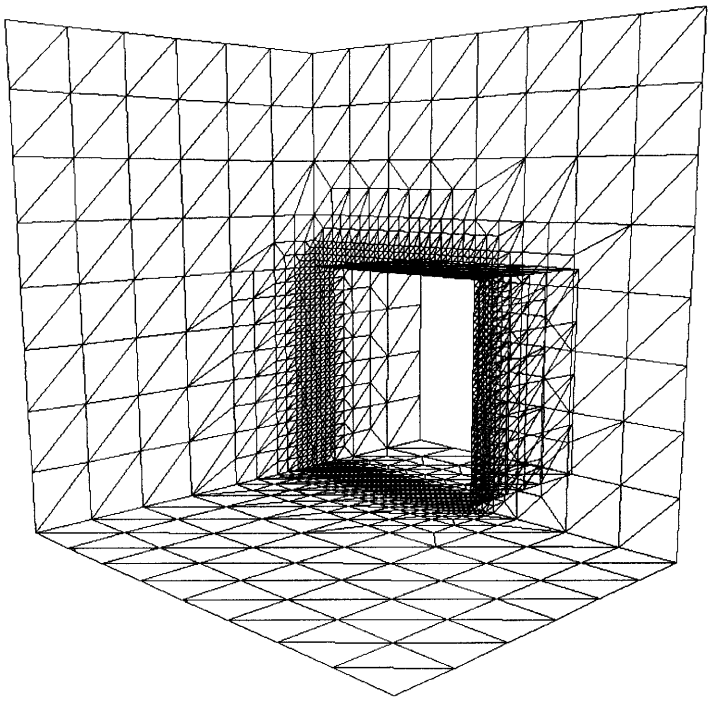

Figure 9. Initial adapted mesh

concentrated over the position of the initial shock plane. A CFL number of 0·75 was used throughout these calculations. The ambient state ahead of the shock has density 1·3 kg m^{-3} and pressure 100,000 Pa, giving an adiabatic sound speed of 328·16 m s^{-1}. We show here results from computations with initial shock speeds of Mach 3 and 1·7, producing post-shock flows which are supersonic and subsonic respectively.

Taking the Mach 3 case first, Figure 10 show the surface of the computational mesh for the Mach 3 calculation at output time, with 'o' marking the origin. This mesh is typical of these calculations, with the regions of mesh refinement clearly marking the position of the active flow field. Here we have specified a maximum of three refined levels. The shock has diffracted outwards through the square box section, with reflective boundary conditions in the $y = 0$ and $z = 0$ planes effectively dividing the computational problem by four. In Figures 11 and 12 we show 35 contours of density through the Mach 3 solution along the cutplane $z = 0{\cdot}2$ (the plane 'bb' in Figure 10) at two different times. In Figure 11 we show the solution at a time of 0·235 ms using *two* levels of refinement in the mesh. In Figure 12 the output time is 0·143 ms, with this time *three* levels of mesh refinement being used. These results indicate that to measurable accuracy at these mesh resolutions at least, the two solutions appear to be self-similar. Some structure in the solution is clear, e.g. the recompression shock surface expected from the 2D calculations,[3] with the $z = 0{\cdot}2$ plane cutting along and across the shock surface, which forms a band around the $x = 0{\cdot}2$ square section entrance. A region of low-density and pressure-separated flow exists 'behind' the corner, with Figures 13 and 14 showing contour plots through the pressure field in the planes $z = 0{\cdot}2$ and $z = 0{\cdot}1$ respectively. In the case of the two-level calculation the initial mesh consisted of 26,929 elements and the final mesh 98,684 elements. For the three-level calculation the initial mesh contained 91,322 elements and the final mesh 267,323 elements at output

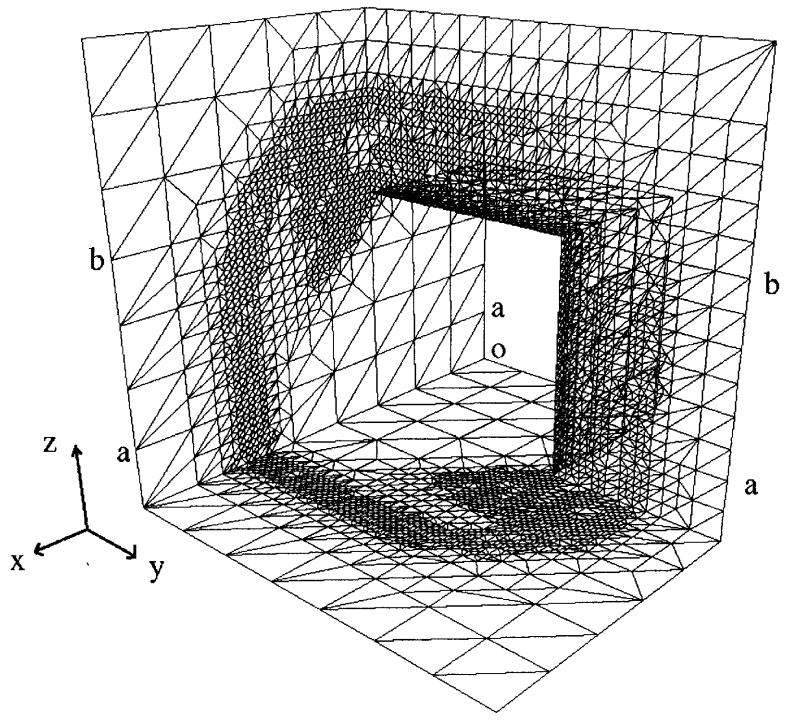

Figure 10. Mach 3 solution adapted mesh

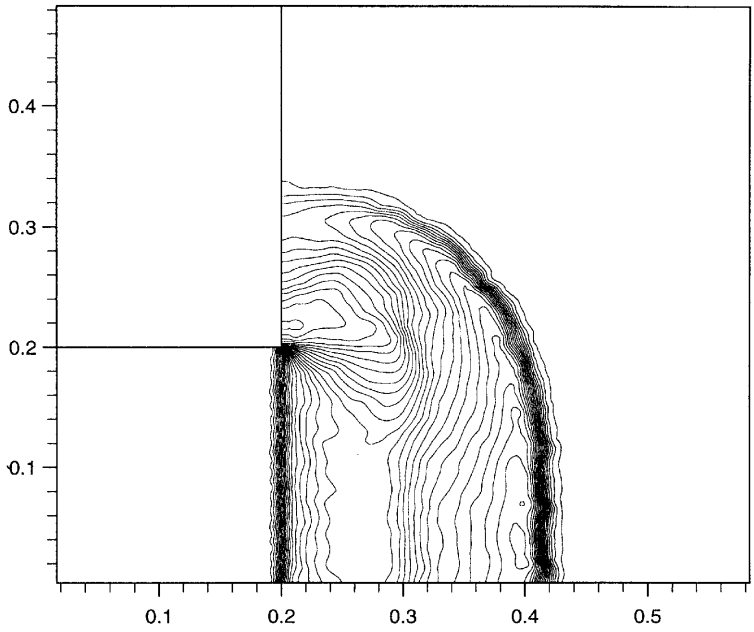

Figure 11. Mach 3 density contours in $z = 0{\cdot}2$ plane with two levels of refinement

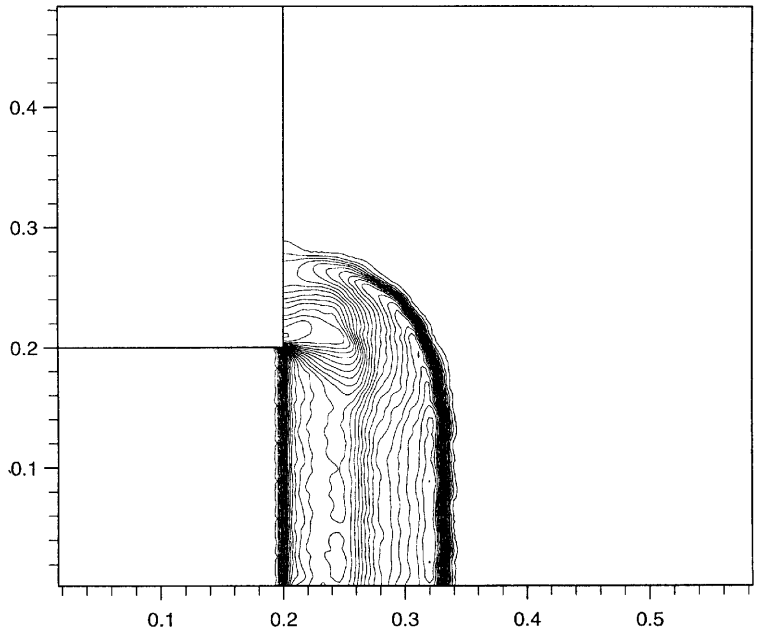

Figure 12. Mach 3 density contours in $z = 0{\cdot}2$ plane with three levels of refinement

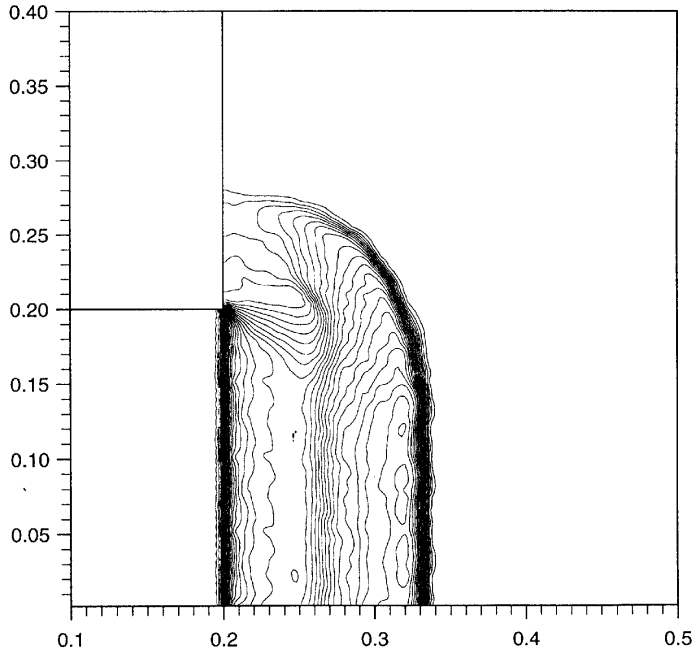

Figure 13. Mach 3 pressure contours in $z = 0{\cdot}2$ plane with three levels of refinement

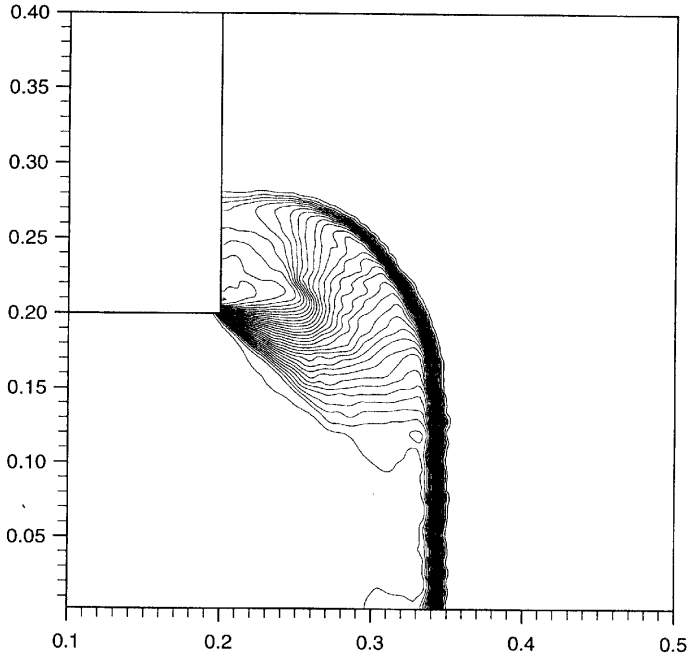

Figure 14. Mach 3 pressure contours in $z = 0{\cdot}1$ plane with three levels of refinement

time. The rate of increase in numbers of elements in the adapted mesh was approximately linear throughout the calculation. These figures should be compared with those of the equivalent 'fixed mesh' (i.e. everywhere fine) of 331,776 elements for the two-level case and 2,654,208 for the three-level case. For the three-level calculation the adaptivity is resulting in a computational speed-up of about a factor of 12·5. Code profiling shows that the overheads associated with the mesh adaptation algorithms are of the order of 15 per cent, with the majority of this currently coming from the edge flagging process, though it is believed that this can be improved upon. Memory management overheads are of the order of 1 per cent.

Looking now at the Mach 1·7 case, Figures 15 and 16 show cutplanes through the density field at $z = 0{\cdot}2$ and $z = 0{\cdot}1$ respectively, on a mesh with three levels of mesh adaption, taken at a time of 0·248 ms. Figures 17 and 18 show the corresponding pressure fields. All the plots are with 35 levels of contouring. The subsonic post-shock flow causes a strong vortex filament to form in the region behind the box section entrance. This can be seen more clearly by the arrow velocity plots for these cutplanes (Figures 19 and 20), obtained from a two-mesh-level calculation at a later time of 0·528 ms. The $z = 0{\cdot}2$ cutplanes show the lengthways cutting of the filament tube, terminating in the oval crossways intersection of the tube with the cutplane in the region of the corner point (0·2, 0·2, 0·2) (see Figure 15 for density and Figure 17 for pressure). This is where the vortex filament rounds the corner of the box section inflow, effectively turning into the plane of the plot. Measurements of the centre of the vortex section taken from the arrow plots in Figure 19 and 20 give the (x, y) co-ordinates of the filament as (0·239, 0·247) in the $z = 0{\cdot}2$ cutplane and (0·224, 0·249) in the $z = 0{\cdot}1$ cutplane at $t = 0{\cdot}528$ ms. On the plane of symmetry at $z = 0$ the vortex centre is placed at (0·225, 0·245). The error in these measurements is difficult to determine, but we estimate it as being of the order of $\pm 0{\cdot}005$ m. These measurements suggest that the filament is advancing faster in the corner than in the region approaching the symmetry plane. The authors believe that recent experimental

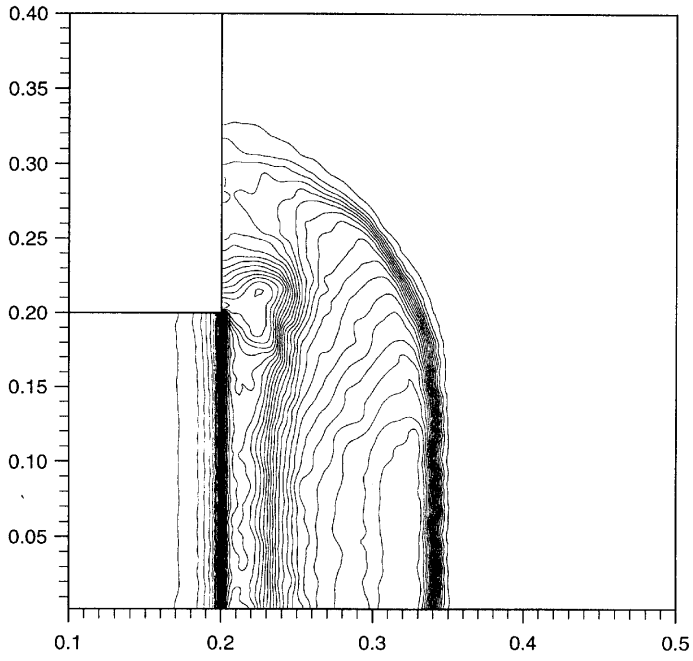

Figure 15. Mach 1·7 density contours in $z = 0{\cdot}2$ plane

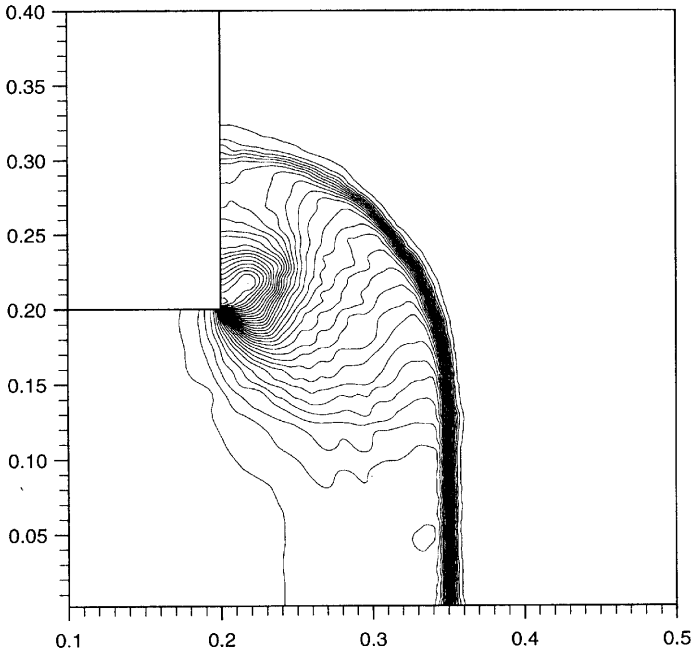

Figure 16. Mach 1·7 density contours in $z = 0{\cdot}1$ plane

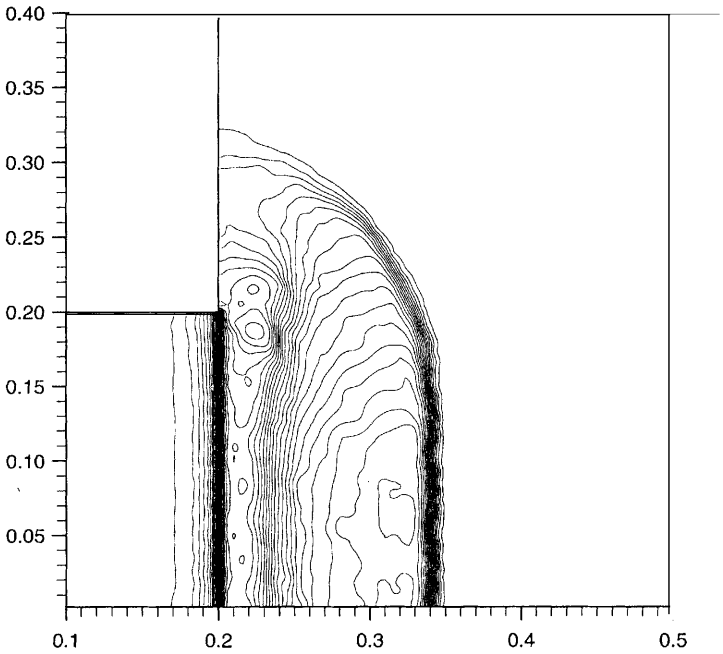

Figure 17. Mach 1·7 pressure contours in $z = 0{\cdot}2$ plane

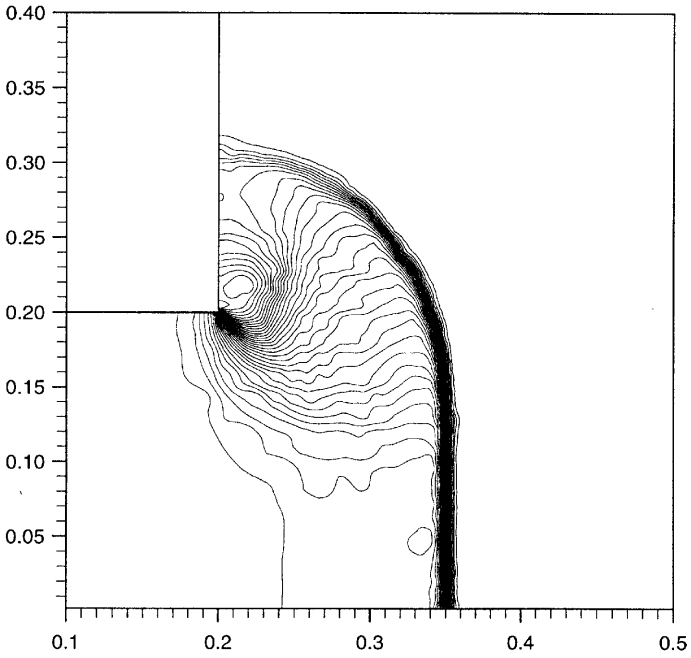

Figure 18. Mach 1·7 pressure contours in $z = 0{\cdot}1$ plane

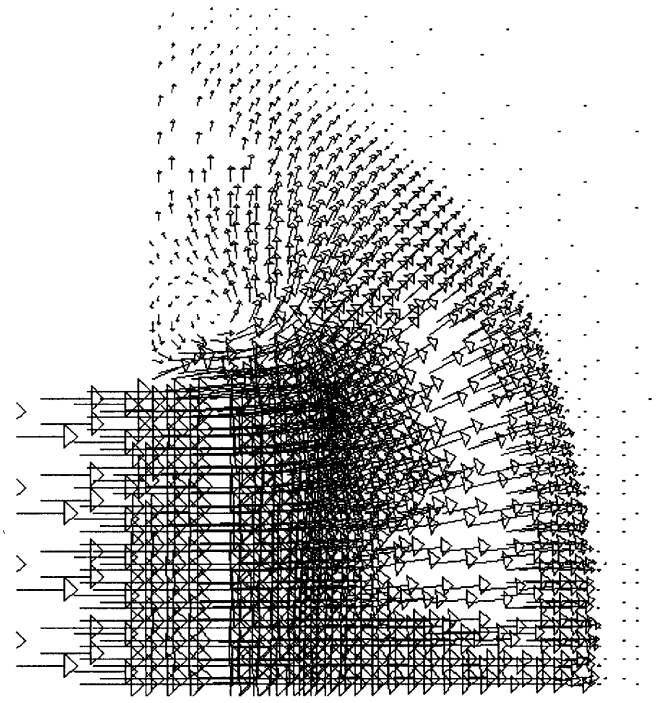

Figure 19. Mach 1·7 velocity vectors in $z = 0{\cdot}2$ plane

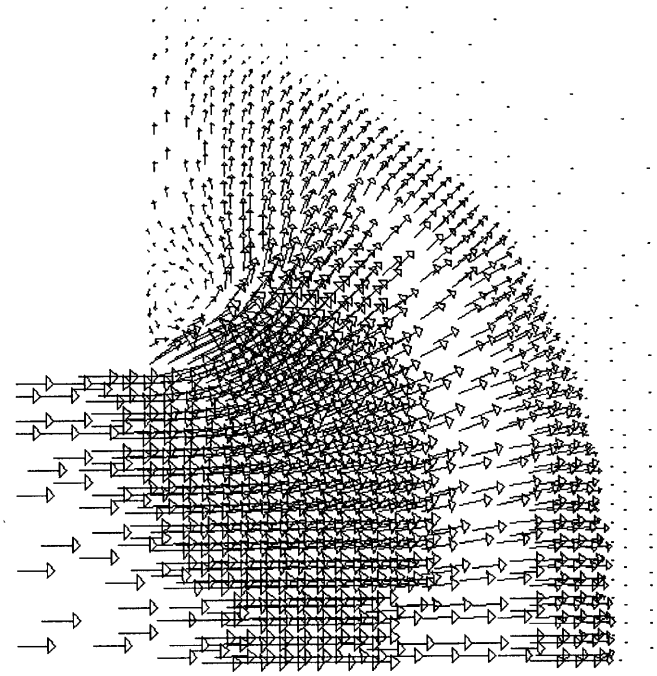

Figure 20. Mach 1·7 velocity vectors in $z = 0{\cdot}1$ plane

results by Takayama's group in Japan have confirmed that the vortex does deform out of the plane of the ring at later times.[25] Careful examination of Figures 15 and 17 shows what appears to be some additional structure in the filament density and pressure sections in the $z = 0{\cdot}2$ plane just adjacent to and below the inflow corner, at the point where the filament curves strongly into the plane. One tentative explanation for this would be a slight kinking in the filament tube in the region of the corner prior to the main tube curvature, which causes a relatively sudden increase in the cross-section of the filament intersected by the $z = 0{\cdot}2$ cutplane. Detailed resolution of the structure within the corner would be an objective of further study. Measurement of the wall shock position in the $z = 0$ symmetry plane gives a wall shock Mach number of 1·15, close to results obtained in two-dimensional calculations.[3] An estimation of the speed of propagation of the ring vortex may be made by comparing the positions of the centre of rotation on the arrow plots at two different times in a given cutplane. Along the $z = 0$ symmetry plane the rotation centre has moved 0·012 m in 0·28 ms, giving an outward ring velocity of about 43 m s^{-1} in the region of the symmetry plane (this is just under half of a previous estimate based on the position of the vortex centre taken at a single time[26]). A more ambitious measurement is to try to estimate the vortical circulation Γ and vortical Mach number $M_\Gamma = \Gamma/2\pi a_\infty$ of the ring. The principal difficulty here is obtaining a reliable measurement of the filament radius. Once this has been determined, then a second-order-accurate interpolant of $\nabla \times \mathbf{V}$ may be constructed (this is actually available as part of the integration algorithm) and the circulation Γ may be found. As a preliminary attempt for the Mach 1·7 vortex, taking a_∞ to be the post-shock sound speed and assuming a filament radius of between 0·01 and 0·02 m, we obtain M_Γ in the filament to be approximately 0·75. Moore's relation[12] between ring velocity and vortical Mach number,

$$U = \frac{\Gamma}{4\pi R}\left(\ln\frac{8R}{a} - \frac{1}{4} - \frac{5}{12}M_\Gamma^2 + O((M_\Gamma)^4)\right),$$

based on an idealized ring vortex structure, predicts a ring velocity of approximately 20–45 m s^{-1} for this M_Γ and filament radius, where we assume a ring radius of 0·2 m. Given the preliminary nature of these measurements as well as the box section geometry, at this stage one can only really conclude that these computational and theoretical results appear to be broadly consistent and indicate a direction for future studies.

The computations were carried out on an SGI eight-processor Power Challenge machine with 256 MB physical memory, with the three-mesh-level calculations taking approximately 900 CPU minutes on a single R4400 processor.

5. CONCLUSIONS AND FURTHER WORK

The results above demonstrate the operation of the mesh adaptation algorithm and solver on a complex time-dependent problem. While they are not yet of sufficient resolution to enable very accurate quantitative measurements to be made on the data, some analysis has been attempted which points the way towards a more detailed numerical study of shock diffraction phenomena. The role of mesh adaptation in the resolution of the small-scale structure in the flow is crucial and the computational speed-up and reduction in overheads associated with the use of these algorithms have been shown to be substantial. In the future it is planned to incorporate the directional refinement capability described above into the code and also to develop robust error estimators to control the mesh adaptation. In addition, a parallel version of the present adaption algorithm is currently under development.[27]

ACKNOWLEDGEMENTS

This work was sponsored by the Pervasive Technology Department at Shell Research Ltd., Thornton Research Centre, U.K. In addition, the first author would like to thank Professor Sam Falle, Dr. Justin Ware and Dr. Paul Batten for some useful discussions and Dr. Paul Selwood for performing the mesh quality experiments.

REFERENCES

1. P. L. Roe, 'Characteristic based schemes for the Euler equations', *Ann. Rev. Fluid Mech.*, **18**, 337–365 (1986).
2. H. M. Glaz, P. Colella, I. I. Glass and R. L. Deschambault, 'A detailed numerical, graphical and experimental study of oblique shock wave reflections', *UTIAS Rep. 285*, University of Toronto, 1986.
3. R. Hillier, 'Computation of shock wave diffraction at a ninety degree convex edge', *Shock Waves*, **1**, 89–98 (1991).
4. P. Colella and L. F. Henderson, 'The Von Neumann paradox for the diffraction of weak shock waves', *J. Fluid Mech.*, **71**, 213 (1990).
5. M. J. Berger and P. Colella, 'Local adaptive mesh refinement for shock hydro-dynamics', *J. Comput. Phys.*, **82**, 64–84 (1989).
6. R. Lohner and J. D. Baum, 'Adaptive h-refinement on 3D unstructured grids for transient problems', *Int. j. numer. methods fluids*, **14**, 1407–1419 (1992).
7. J. Ware, 'The adaptive solutions of time-dependent partial differential equations in two-dimensions', *Ph.D. Thesis*, University of Leeds, 1993.
8. R. Biswas and R. C. Strawn, 'A new procedure for dynamic adaption of 3-D unstructured grids', *Appl. Numer. Math.*, **13**, 437–452 (1994).
9. Y. Kallinderis, V. Parthasarathy and J. Wu, 'A new Euler scheme and adaptive refinement/coarsening algorithm for tetrahedral grids', *AIAA Paper 92-0446*, 1992.
10. B. van Leer, 'On the relationship between upwind difference schemes', *SIAM J. Sci. Stat. Comput.*, **5**, (1984).
11. E. F. Toro, M. Spruce and W. Speares, 'The restoration of the contact surface in the HLL-Riemann solver', *Shock Waves*, **4**, 25–34 (1994).
12. D. W. Moore, 'The effects of compressibility on the speed of propagation of a vortex ring', *Proc. R. Soc. Lond. A*, **397**, 87–97 (1987).

13. S. K. Godunov, 'A finite difference method for the numerical computation of discontinuous solutions of the equations of fluid dynamics', *Mat. Sb.*, **47**, 357–393 (1959).
14. P. Batten and D. M. Causon, 'Positively conservative high resolution convection schemes for unstructured elements', *Int. j. numer. methods eng.*, submitted.
15. A. Harten, P. D. Lax and B. van Leer, 'On upstream differencing and Godunov type schemes for hyperbolic conservation laws', *SIAM Rev.*, **25**, 36–61 (1983).
16. E. F. Toro, 'A linearized Riemann solver for the time-dependent Euler equations of gas dynamics', *Proc. R. Soc. Lond. A*, **434**, 683–693 (1991).
17. W. Speares and E. F. Toro, 'A high resolution algorithm for shock dominated problems with adaptive mesh refinement', *ZFW J. Flight Sci.*, **19**, 267–281 (1995).
18. M. Berzins, J. Ware and J. Lawson, 'Spatial and temporal error control in the adaptive solution of systems of conservation laws', in *Advances in Computational Methods for PDEs*, *IMACS PDE VII*, IMACS, 1992.
19. M. E. G. Ong, 'Uniform refinement of tetrahedron', *SIAM J. Sci. Comput.*, **15**, 5 (1994).
20. R. E. Bank, *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations: User's Guide 7.0*, SIAM, Philadelphia, PA, 1994.
21. A. Liu and B. Joe, 'Quality local refinement of tetrahedral meshes based on 8-subtetrahedron subdivision', *Preprint*, University of Alberta, 1994.
22. N. Weatherill, personal communication, 1994.
23. R. Biswas and R. C. Strawn, 'Mesh quality control for multiply-refined tetrahedral grids', *Appl. Numer. Math.*, in press.
24. P. Batten, C. Lambert, E. F. Toro, R. Saunders and D. M. Causon, 'A temporal refinement algorithm for unstructured mesh methods', *Proc. ICFD Conf.*, Oxford, April 1995.
25. H. Babinsky, personal communication, 1995.
26. W. Speares and M. Berzins, 'A fast 3D unstructured mesh adaption algorithm with time dependent upwind Euler shock diffraction calculations', *Proc. 6th Int. Symp. on CFD*, Lake Tahoe, NV, 1995, Vol. 3, p. 1181.
27. D. Hodgson, P. K. Jimak, P. Selwood and M. Berzins, 'Scalable parallel generation of partitioned unstructured meshes', *Proc. CFD '95 Conf.*, Pasadena, CA, 1995.
28. A. Harten, 'High resolution schemes for hyperbolic conservation laws', *J. Comput. Phys.*, **78**, 437–458 (1983).
29. Strang, 'On the construction and comparison of difference schemes', *SIAM J. Numer. Anal.*, **5**, 506–517 (1968).
30. B. van Leer, 'Towards the ultimate conservative difference scheme V', *J. Comput. Phys.*, **32**, 101–136 (1979).

