A Cartesian Grid Approach with Hierarchical Refinement for Compressible Flows

James J Quirk[1]
Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA 23681, USA.

ABSTRACT

Many numerical studies of flows that involve complex geometries are limited by the difficulties in generating suitable grids. We present a Cartesian boundary scheme for two-dimensional, compressible flows which is unfettered by the need to generate a computational grid and so it may be used, routinely, even for the most awkward of geometries. In essence, an arbitrary-shaped body is allowed to blank out some region of a background Cartesian mesh and the resultant cut-cells are singled out for special treatment. This is done within a finite-volume framework and so, in principle, any explicit flux-based integration scheme can take advantage of this method for enforcing solid boundary conditions. For best effect, the present Cartesian boundary scheme has been combined with a sophisticated, local mesh refinement scheme, and a number of examples are shown in order to demonstrate the efficacy of the combined algorithm for simulations of shock interaction phenomena.

[1]This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-19480 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681.

1 INTRODUCTION

Of the three basic strategies that have been employed to compute flows with complex geometries, the Cartesian boundary approach has received the least attention; in contrast, both the unstructured mesh approach (*e.g.* [1] and the composite body-fitted grid approach (*e.g.* [2]) have large followings. This lack of attention is surprising given its conceptual simplicity. Admittedly, a number of obstacles have to be overcome so as to produce a working scheme, but this is also true of the other two approaches. For example, it is very difficult to automate the process of generating composite grids for genuinely complex geometries, and the resultant inter-grid boundaries complicate the method of flow solution[3]. Similarly, there is evidence to suggest that the unstructured grid approach is slightly at odds with the requirements of the flow solver. For example, for strong shock waves, unstructured grid schemes suffer larger phase errors than do structured grid schemes[4].

In this paper, we present a general purpose Cartesian boundary method for computing shock interactions that involve complex geometries. It will become clear that this method relies more on sophisticated logic than on sophisticated mathematics. Indeed, the biggest drawback of the Cartesian boundary approach, and one which will always act to limit its following, is the fact that there is no concise recipe. The method relies on being able to handle exceptions and is therefore much more verbose than say an unstructured grid method. In part, this explains why most Cartesian schemes only work for stylized geometries where the necessary logic is greatly reduced and the development costs are low. The strength of the present method lies in its ability to cope with truly arbitrary geometries.

Space does not permit us to provide an adequate survey of existing Cartesian boundary schemes, and so the following references, whilst not completely exhaustive, should suffice to indicate research activity in this area[5]–[19]. Where appropriate, direct references will be made to some of these works in the main text. Moreover, since a detailed description of our scheme has already appeared in the literature[17], here we only elaborate on those aspects of the scheme which appear to have caused some confusion. Therefore we recommend that this paper be read in conjunction with the original article so that it does not appear disjointed.

The rest of this paper is as follows. In the next section, we outline certain components of our Cartesian boundary scheme, and we endeavour to reveal the obstacles that shaped them. For practical purposes, any Cartesian boundary scheme must be combined with some form of local mesh refinement. Otherwise, the background

mesh would in general require an inordinate number of cells just to unambiguously determine the input geometry. In Section 3, we present our preferred form of mesh refinement – the Adaptive Mesh Refinement (AMR) algorithm[20]. Following which, in Section 4, examples are given to demonstrate the efficacy of the present combined Cartesian boundary-mesh refinement scheme for investigating shock interaction phenomena. Finally, in Section 5 some conclusions are drawn concerning Cartesian boundary schemes.

2 CARTESIAN BOUNDARY SCHEME

We first reconsider the seemingly innocuous problem of determining which cells of the background mesh are blanked out by the input geometry. Then we re-examine the method by which we overcome the stability problems due to the presence of arbitrarily small cut-cells. Finally, we outline how our method can be extended to cope with moving bodies

2.1 Geometric Considerations

The first step in any Cartesian boundary scheme is to determine which mesh cells lie inside, outside or on the solid boundaries specified as input. The sophistication of this step will largely determine the performance of the overall algorithm. The simplest strategy is to approximate the boundaries by a series of steps, thus there are only two types of cells: solid cells which lie wholly inside a body, and uncut cells which lie wholly outside a body. Unfortunately, this simple strategy does not work well in general, because the corrugations along the approximation to a curved boundary will inevitably cause acoustic disturbances which pollute the flow solution. However, Falle & Giddings[10] have shown that the introduction of some viscosity can restrict such disturbances to a narrow boundary layer, and so this method should not be rejected out of hand. We elected to allow cut-cells, thus solid boundaries are approximated by a series of straight line segments. This approach requires us to find the actual intersection points between the background grid and the input geometry, by tracing its outline. Superficially this task seems straightforward. But, if due care is not taken, round-off errors will cause problems such as an intersection point being missed or duplicated.

Although such problems are rare, a *robust* scheme must prevent them from *ever* happening or at least ensure that nothing untoward occurs as a result. We elected to

dispense with round-off error altogether and developed a procedure that would find the intersection points relative to a discrete lattice using only exact operations[17]. Thus complete control is exercised over the process of determining the intersection points and so no point can ever be missed or duplicated. On the other hand, Rice[21] attempted to overcome round-off problems by basically employing tolerances when making floating point comparisons. This *solution* results in too many restrictions on the input geometry (see [21]) and, in our opinion, is inelegant. It may even be slightly dangerous in that it is not machine independent. For example, consider what might happen if the intersection points are found using a heterogeneous parallel computing system. If an intersection lies in the vicinity of a processor partition boundary, it is conceivable that only one of the affected nodes will find the intersection and so there will be an inconsistency. Admittedly, corrective action could be taken by some fix-up procedure, but this would introduce the unnecessary overhead of inter-processor communication. In general, it is far better to circumvent problems than to attempt to cure them when they occur.

Once all the intersection points are found, they must be collated so as to determine the nature of the cut cells. For simplicity, we elected to handle only the three basic types of cell formed from the intersection of a single straight line segment, which together with the four possible orientations gives the twelve types of cut-cell shown in Figure 1. Note that we do not allow corners to occur within a cell.

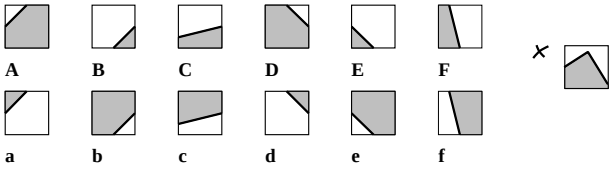


Figure 1: Basic types of cut-cell.

Since there is no limit to the number of intersections that might occur for a given cell, its type is generally determined from its first and last intersection points as shown in Figure 2. Under normal circumstances, a cell having more than two intersection points merely indicates that the mesh is too coarse to resolve the input geometry properly, in which case, we locally refine the mesh so as to get a better representation of the boundary.

In certain circumstances, say near cusps, some cells are found to be degenerate and a blunting procedure is applied in order to remove the degenerate cells from the boundary representation, see Figure 3.1. Here the degree of blunting is excessive and

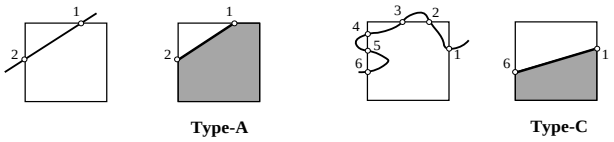


Figure 2: Collation of intersection points.

could be reduced by introducing further cell-types but this would further complicate what is already a fairly busy scheme. Instead, as shown in Figure 3.2, we employ local mesh refinement to reduce the blunting to an acceptable level.

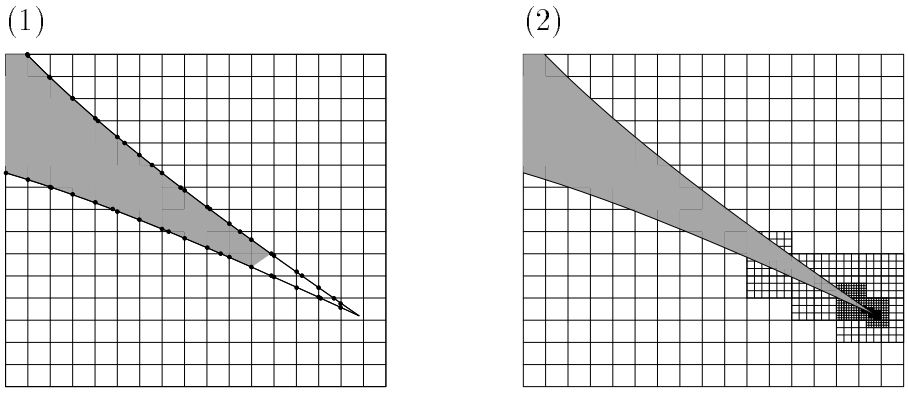


Figure 3: Local mesh refinement is used to control the blunting of sharp corners.

This blunting procedure has come in for some criticism since it is perceived to alter the input geometry[22]. But, if it is used in conjunction with local mesh refinement, any alterations are on a scale so small as to be masked by the inherent dissipation of a shock-capturing scheme. In effect, numerical diffusion results in a small separation bubble to round off any singularity in the input geometry. Thus our blunting procedure, if used sensibly, has minimal affect on the flow solution, and results given in Section 4 substantiate this claim. Besides, at a more philosophical level, one could argue that if such alterations did matter, no simulation could ever hope to reproduce an experiment since no very sharp corner is precise in its manufacture. But this runs against common experience and so *imperceptible* alterations do not matter: any discretization is but an approximation to the input geometry.

As will be shown in Figure 11, our two-dimensional Cartesian boundary scheme is able to handle arbitrary geometries, automatically. Yet we have not attempted to extend the method to three-dimensions, simply because the task of determining the

cut-cell types will dominate proceedings, and our interests are of a more fluid dynamical nature. The simplest strategy would be to produce some surface triangulation of the object of interest and compute the intersections with the Cartesian mesh triangle by triangle. But unless the triangles are much smaller than the smallest mesh cell used for the computation, this strategy will prove unsatisfactory because one will just resolve the triangular facets and not the true surface geometry. Moreover, with a local refinement scheme it may not be possible to predict ahead of time how small the smallest cell will be. Melton *et al.*[14] have adopted the only sensible approach and are using a commercial CAD package to provide the correct surface representation. However, such packages are usually proprietary and are therefore difficult to obtain for research purposes.

Whereas Melton *et al.* are using a surface representation and are laboriously developing the machinery to compute the grid intersection points themselves, we would advocate using a solid modeller based on a Polygonal-Map octree[23]. Such modellers could provide the cut-cell information directly. In effect, they represent an object by a number of cuboidal elements, maintaining the precise surface geometry of each element. If the elements were made small enough, say to match the size of mesh cell needed for a fluids computation, the nature of most cut-cells would follow immediately. Although, a blunting procedure might have to be applied so as to remove certain degenerate elements as is done in two-dimensions. Given such a package the extension of our Cartesian boundary scheme to three-dimensions would be straightforward.

2.2 Stability Considerations

Since cut-cells can be arbitrarily small, a Cartesian boundary scheme must address the stability problems caused by having disparate cell sizes. For steady-state computations, De Zeeuw & Powell[19] have demonstrated that straightforward local time-stepping is sufficient to ensure stability. On the other hand, unsteady flow computations require a more sophisticated strategy. For example, Berger & Le Veque[6] utilized a large time-step generalization of Godunov's method which keeps track of individual waves as they move across the mesh. This scheme does not suffer an explicit restriction on the size of stable time step and so very small cut cells can be safely integrated at the time step used to integrate uncut cells. As an alternative, Pember *et al.*[15] redistribute part of the computed updates for small cut cells to neighbouring cells, following certain rules which ensure stability. As yet another alternative, we

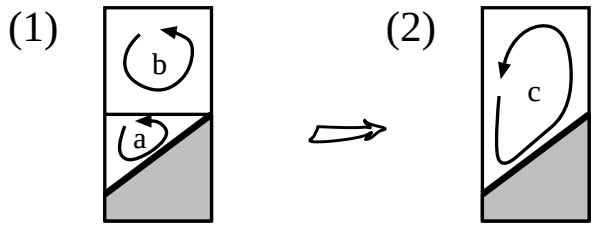


Figure 4: A merging strategy is used to remove small cells.

employ a cell merging technique which is a generalization of the method employed by both Clarke *et al.*[8] and by Chiang *et al.* [7]. Ultimately, whatever method is chosen, it must work in the most general of cases, otherwise it negates the principal motivation for developing a Cartesian boundary scheme: the promise of being able to handle arbitrary geometries in a completely automatic fashion.

To see how our approach works in the simplest case, consider Figure 4.1. Suppose an update is computed for each cell using a one step finite-volume scheme. The updates to the conserved variables, $\Delta\mathbf{W}_a$ and $\Delta\mathbf{W}_b$, may be written

$$\Delta\mathbf{W}_a = -\frac{\Delta t}{V_a}\sum_{\text{faces } a}\mathbf{F}.\mathbf{A} \quad \text{and } \Delta\mathbf{W}_b = -\frac{\Delta t}{V_b}\sum_{\text{faces } b}\mathbf{F}.\mathbf{A} \ ,$$

where V_a and V_b are the volumes of the cells, and $\mathbf{F}$ is the flux acting through the face $\mathbf{A}$. If the time step Δt is based on the size of the uncut cell b, the solution within a will be unstable. To ensure stability, the updates for the two cells are replaced by some fraction of their volume-weighted average. Since the volume weighted average is equivalent to the update that would have been computed for the composite cell shown in Figure 4.2,

$$\frac{V_a\Delta\mathbf{W}_a + V_b\Delta\mathbf{W}_a}{V_a + V_b} \quad = \quad -\frac{\Delta t}{V_c}\sum_{\text{faces } c}\mathbf{F}.\mathbf{A},$$

the appropriate fractions are $\frac{V_a}{(V_a+V_b)}$ for cell a and $\frac{V_b}{(V_a+V_b)}$ for cell b. Thus, effectively we would have a grid that contains the cell c instead of the two cells a and b. Although this merging process inevitably reduces the accuracy of the integration scheme at solid boundaries, Coirier & Powell[9] have shown that it does not affect the global accuracy. Also, if needs be, the local loss in accuracy can be recovered using mesh refinement.

The generalization of this method rests on finding a set of lists, where each list identifies a group of cells that need to be merged together so that certain small cut-

cells do not cause instabilities, see [17] for the details. Note that a list can contain several cells, but no one cell appears in more than one list. The cut-off point for determining whether a cell is small or not is completely arbitrary. In practice, we have found that cells larger than half the size of an uncut cell do not cause problems and so are not deemed to be small. Note that our procedure is just a convenient method for computing updates for awkward shaped cells from a small number of fixed cell-types for which the update is well defined and easily coded.

For example, a type-A cell has just three sides. The flow solution in such a cell can be reconstructed using the method proposed by De Zeeuw & Powell[19]. Following which, it is a straightforward matter to compute the three fluxes acting on these faces, using one's favourite upwind scheme. Note, as is common practice, the flux for the boundary face is computed by reflecting the normal momentum at the wall. The cell-update then follows trivially to be used later on by the cell merging procedure. Given that no one cell appears in more than one combination list, our integration procedure is conservative.

2.3 Extension to Moving Bodies

The next logical step in the development of our Cartesian boundary scheme is its extension to moving bodies. Like most components of the scheme this extension is simple in concept, but awkward to implement in a foolproof manner, and our own efforts have been stymied by other research commitments. Nevertheless, we outline the strategy that we have devised[24] and note that it is basically the same as that devised by Bayyuk *et al.*[5]. Whilst the strategy is clear, certain implementation details need to be ironed out.

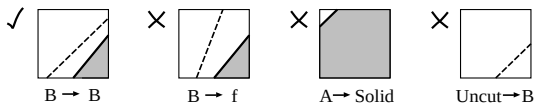
Consider a body which is moving relative to a background Cartesian mesh, say in a north westerly direction. Figure 5 shows some of the changes that a cell might undergo during a time step from t^n to t^{n+1}. If the cell has the same type at the end of the time step as it did at the start, the cell may be integrated trivially using the following finite-volume discretization

$$V^{n+1}\mathbf{W}^{n+1} = V^n\mathbf{W}^n - \Delta t \sum_{faces} \mathbf{F}^n.\mathbf{A} - (0,0,0,\bar{p})^t(V^{n+1} - V^n).$$

Here V^n and V^{n+1} are the volumes of the cell at the start and end of the time step Δt. $\mathbf{W}$ is the conserved variable vector per unit volume, and $\mathbf{F}^n$ is the flux through a face whose average area is $\mathbf{A}$ over the course of the time step. Similarly, p is the average pressure which acts on the solid boundary and so the last term is effectively

the work done by the boundary displacing a volume of fluid ($V^{n+1} - V^n$). Difficulties only arise if the cell changes type during the time step as is the case for three of the examples in Figure 5.1.

(1)



(2)

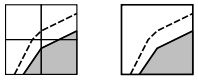


Figure 5: If a body moves, individual cells may change type.

The solution *trick* is to find groups of cells such that the type for the group remains constant over the time step as shown in Figure 5.2. Then the above finite-volume discretization may be applied straightforwardly to the composite cell. To see how this may be implementated in the general case consider Figure 6. Figure 6.1 shows the outline of some body at the start and end of a time step. Figure 6.2 shows two curves C_1 and C_2 which are the external hulls of those cells which are cut at either t^n or t^{n+1}. If the body is non-deformable, these curves cannot cross. The problem of finding suitable combination groups is reduced to connecting up C_1 and C_2 along the co-ordinate lines as shown in Figure 6.3. In this case, Figure 6.4 shows the resultant groups. Some of these combination groups may then have to be merged with other cells, as in the previous section, to ensure stability.

Although the above procedure is straightforward, it has proven difficult to code in a manner that matches the generality of the rest of the algorithm. Moreover, it has certain inherent limitations that some may find objectionable. For example, the procedure to find the combination groups is not unique. Consider the case where a planar piston is moving at 45° to the mesh, see Figure 7. If care is not taken, the combination groups could alternate between running vertically and running horizontally. This would result in information propagating along the face of the piston at non-physical speeds. Bayyuk *et al.*[5] identify some other weaknesses.

Although the extension to moving bodies clearly has some weaknesses, the early results are encouraging and we feel this approach is worth persuing, especially given the the exciting new applications that it would open up.

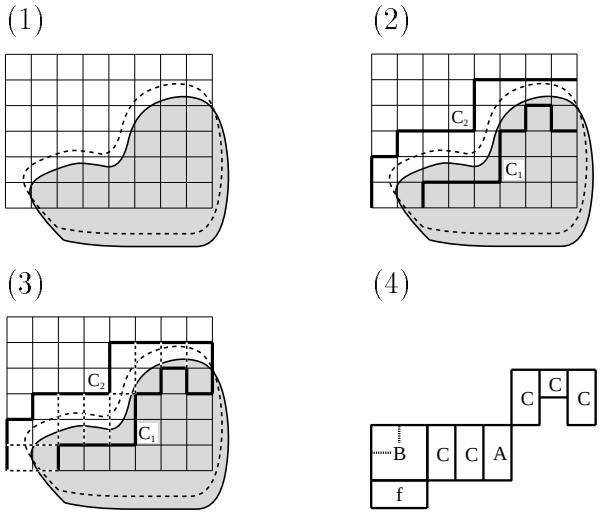


Figure 6: Strategy for finding groups of cells whose type remains constant when a body moves.

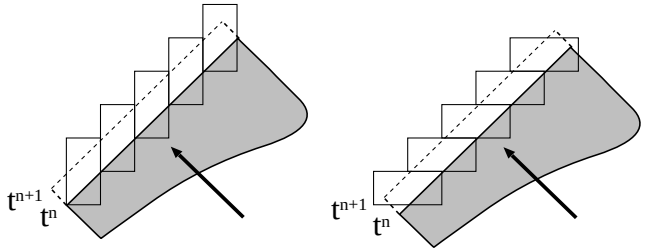


Figure 7: Problems could arise, if combination cells alternate in orientation.

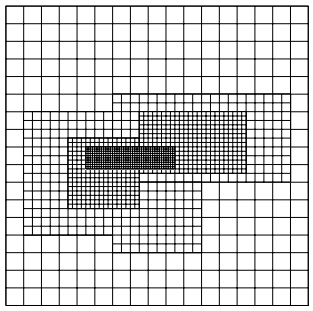
3 THE AMR ALGORITHM

The Adaptive Mesh Refinement (AMR) algorithm is a general purpose scheme for integrating systems of hyperbolic partial differential equations. It attempts to reduce the costs of integration by matching the local resolution of the computational grid to the local requirements of the solution being sought. The foundations of the algorithm lie with the work of Berger & Colella[25], but the derivative outlined here is due to Quirk[20].

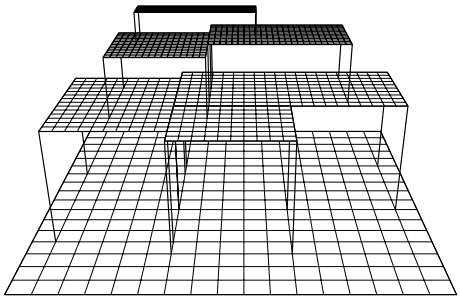
The AMR algorithm employs a hierarchical grid system. In the following, the term 'mesh' refers to a single topologically rectangular patch of cells and the term 'grid' refers to a collection of such patches. At the bottom of the hierarchy a set of coarse mesh patches delineates the computational domain. These patches form the

grid G_0 and they are restricted such that there is continuity of grid lines between neighbouring patches. This domain may be refined locally by embedding finer mesh patches into the coarse grid G_0. These embedded patches form the next grid in the hierarchy, G_1. Each embedded patch is effectively formed by subdividing the coarse cells of the patches that it overlaps. The choice for the refinement ratio is arbitrary, but it must be the same for all the embedded patches. Thus, by construction, the grid G_1 also has continuity of grid lines. This process of adding grid tiers to effect local refinement may be repeated as often as desired, see Figure 8.

From stability considerations, many numerical schemes have a restriction on the size of time step that may be used to integrate a system of equations. The finer the mesh, the smaller the allowable time step. Consequently, the AMR algorithm refines in time as well as space. More but smaller time steps are taken on fine grids than on coarse grids in a fashion which ensures that the rate at which waves move relative to the mesh (the Courant number) is comparable for all grid levels. This avoids the undesirable situation where coarse grids are integrated at very small Courant numbers given the time step set by the finest grid's stability constraints.



plan view



perspective view

Figure 8: The AMR algorithm employs a hierarchical grid system.

The field solution on each grid is retained even in regions of grid overlap and so all grid levels in the hierarchy coexist. The order of integration is always from coarse to fine since it is necessary to interpolate a coarse grid solution in both time and space to provide boundary conditions for its overlying fine grid. The various integrations at the different grid levels are recursively interleaved to minimize the span over which any temporal interpolation need take place. Periodically, for consistency purposes, it is necessary to project a fine grid solution on to its underlying coarse grid. Figure 9 shows the sequence of integration steps and back projections for a three level grid $\{G_0, G_1, G_2\}$ with refinement ratios of 2 and 4.

INTEGRATION	TIME STEP	PROJECTION	ADAPTION
G_0	Δt		
G_1	$\Delta t/2$		
$4 \times G_2$	$4 \times \Delta t/8$		
		$G_2 \rightarrow G_1$	
			G_2
G_1	$\Delta t/2$		
$4 \times G_2$	$4 \times \Delta t/8$		
		$G_2 \rightarrow G_1$	
		$G_1 \rightarrow G_0$	
			G_2
			G_1

Figure 9: Grid operations are recursively interleaved (to be read from top to bottom).

The integration of an individual grid is extremely simple in concept. Each mesh is surrounded by borders of dummy cells. Prior to integrating a grid, the dummy cells for every mesh patch in the grid are primed with data which is consistent with the various boundary conditions that have to be met. Each mesh patch is then integrated independently by an application dependent, black-box integrator that never actually sees a mesh boundary. Thus, in principle, any cell-centred scheme developed for a single topologically rectangular mesh can form the basis for the integration process.

In general it is necessary to adapt the computational grid to the changes in the evolving flow solution and so the grid structure is dynamic in nature. Monitor functions based on the local solution are used to determine automatically where refinement needs to take place so as to resolve small scale phenomena[20]. For example, Figure 10 shows several snapshots taken from the simulation of a shock wave diffracting around

a corner. Each snapshot shows the outlines of the mesh patches which go to make the finest grid. This grid clearly conforms to the main features of the flow, namely the diffracted shock front and the vortex located at the apex of the corner. Although the changes in grid structure shown here are dramatic, many adaptions have taken place between each frame. A large number of small grid movements occurs because the adaption process dovetails with the integrations process, see Figure 9. Note that the adaption always proceeds from fine to coarse so as to ensure that there is never a drop of more than one grid level at the edge of a fine grid to the underlying coarse grid. A grid adaption essentially produces a new set of mesh patches which must be primed with data from the old set of patches before the integration process can proceed. Where a new patch partially overlaps an old patch of the same grid level, for the region of overlap, data may be simply shovelled from the old patch to the new patch. In regions of no such overlap, the required field solution is found by interpolation from the underlying coarse grid solution.

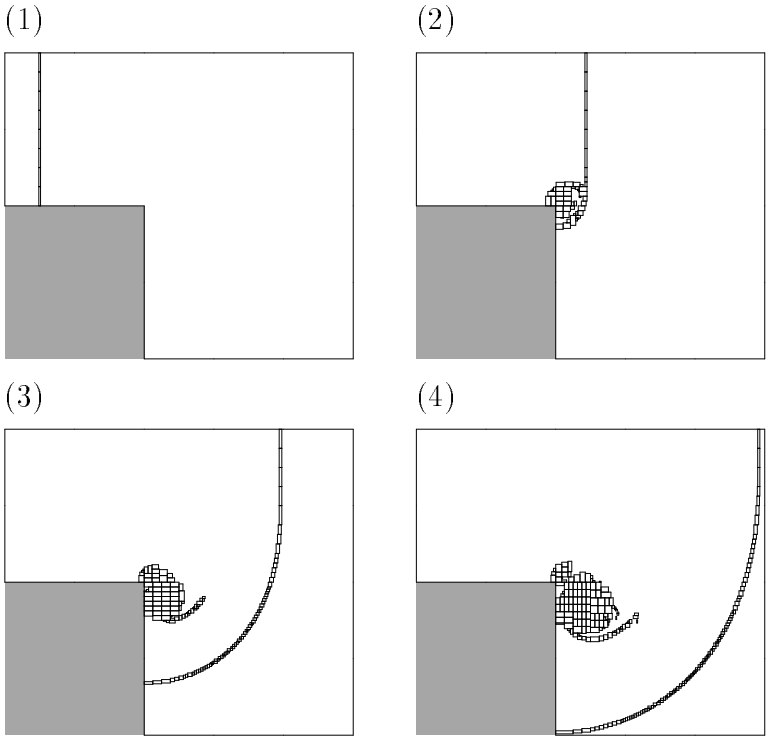


Figure 10: The AMR algorithm employs a dynamic grid system.

In a typical application the finest grid will contain several hundred mesh patches. Thus, the mesh patch is a sufficiently fine unit of data for efficient parallelism. The parallel AMR algorithm[26] is implemented using a Single Program Multiple Data

(SPMD) model. Each processing node executes the basic serial algorithm[20] in isolation from all other nodes, except that at a few key points messages are sent between the nodes to supply information that an individual node deems to be missing, that is off-processor. For example, during the integration of a grid, the only point at which a processor needs to know about other processors is during the priming of the dummy cells. Whereas in a serial computation all data fetches are from memory, for a parallel computation some are from memory and some necessitate receiving a message from another processor. Each time the grid adapts, the algorithm generates a schedule of tasks that have to be performed so as to prime correctly the dummy cells of a given grid. If running in parallel, this schedule is parsed to produce a schedule of those tasks that necessitate off-processor fetches. At which point, individual processors can exchange subsets of their fetch schedules, as appropriate, so that every node can construct a schedule of messages that it must send out at some later date. Thus, the priming process is carried out in two phases. First, all the local data fetches are performed as for the serial case. Second, each node sends out the data that has been requested of it. The node then waits for those data items it has requested. For each incoming message it can readily determine from its own schedules what to do with the off-processor data, and so the order in which messages arrive is unimportant. The adaption process and the back projection of the field solution between grid levels also necessitate sizeable amounts of communication, these are handled in a similar fashion to the priming of the dummy cells.

The problem of load balancing the AMR algorithm rests on determining the best distribution of the new patches amongst the processing nodes before the new field solution is interpolated from the old field solution. Currently, this is done using heuristic procedures[27] which bear strong similarities to classical 'Bin Packing' algorithms[28] with the added complication that they must account for the communication costs of data transfer between nodes.

The main advantage of the AMR algorithm is that the processing within a patch can proceed largely without knowledge of the method of parallelization or knowledge of the treatment of mesh boundaries, and so it is extremely simple to change the basic method of flow integration. Thus the present Cartesian boundary scheme can utilize the algorithm more or less directly. Except that there is a small amount of additional book keeping to account for the fact that some groups of combination cells may straddle more than one processor. But this complication is not great and introduces very little data traffic.

4 RESULTS

All the simulations reported in this section were done by integrating the Euler equations using the present Cartesian boundary scheme in conjunction with the finite-volume method described in [17]. Each computation was performed in parallel on a cluster of five Silicon Graphics workstations (Indigo 2, MIPS 4400).

In order to demonstrate that our scheme can cope with arbitrary two-dimensional geometries, we have computed the interaction of a planar shock wave, $M_S = 1.5$ and $\gamma = 1.4$, with the logo 'AMR @ ECCOMAS 94'. Although this example is unashamedly gratuitous it serves to demonstrate the capabilities of the scheme. The whole exercise from conception to completion took just seven hours and involved no special intervention on our part. Figure 11 shows a schlieren-type snapshot from this simulation. The background Cartesian mesh was nominally equivalent to a uniform mesh of 1920 by 600 cells and so the flow field is well resolved and many fundamental shock interaction phenomena are clearly visible.

Whilst spectacular, given the impossibility of verifying the results, this simulation is rather meaningless. Therefore, on a more serious note, we present two schlieren-type images from a simulation of the focusing of a weak shock wave, $M_S = 1.2$ and $\gamma = 1.4$, by a parabolic reflector, see Figure 12. These images compare well with experiment (see, Figures 3 (a) and 3 (f) of [29]) and so the integrity of the simulation is beyond doubt. In this case, although the geometry is relatively simple, a topologically uniform body-fitted grid would be severely distorted. Since such distortions could have an adverse affect on the quality of the simulation, it follows that a Cartesian boundary scheme need not be reserved for geometrically complex problems.

To investigate the potential vagaries of the blunting procedure which is applied to sharp corners, we have simulated the diffraction over a knife edge of a $M_S = 1.5$ planar shock wave. This flow gives rise to a vortex sheet which emanates from the tip of the knife edge[30]. Figure 13 shows a sequence of schlieren-type images for various stages in the development of the vortex sheet. Frames 1–5 were taken from a computation for which the knife edge was blunted. The computation was then repeated with the knife edge positioned so that it was not blunted, see Figure 14. Qualitatively, the differences in the two solutions are minor; c.f Frame 5 (with blunting) and Frame 6 (without blunting). Generally speaking, a fluid dynamicist would be more concerned about the validity of simulating a viscous phenomena inviscidly. Consequently, although the solution is sound, one should be careful in attaching too much

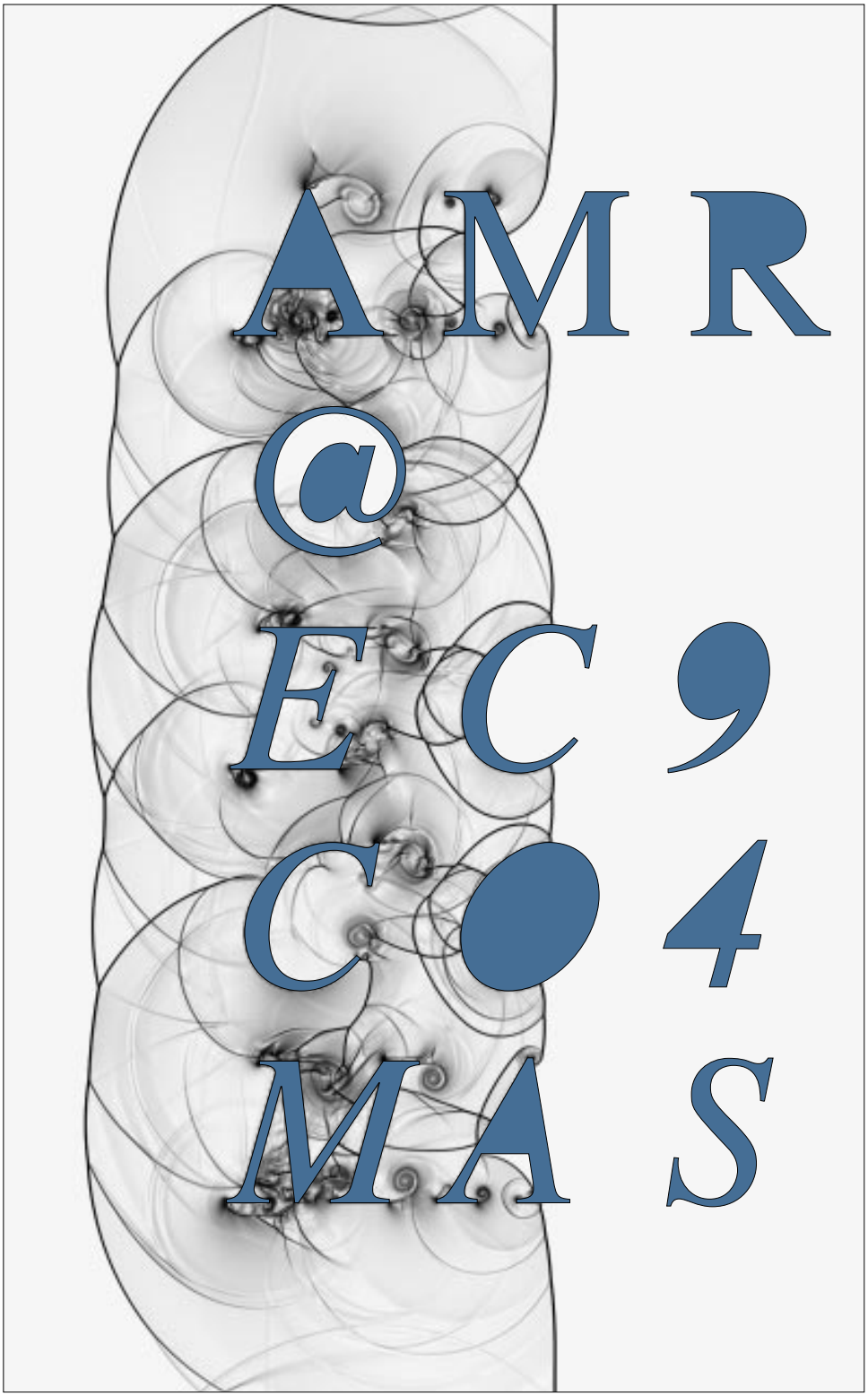


Figure 11: The algorithm can cope with arbitrary geometries: flow around 'AMR @ ECCOMAS 94'!

plots/focus1.ps

plots/focus2.ps

Figure 12: Two schlieren-type snapshots from the focusing of a weak shock wave.

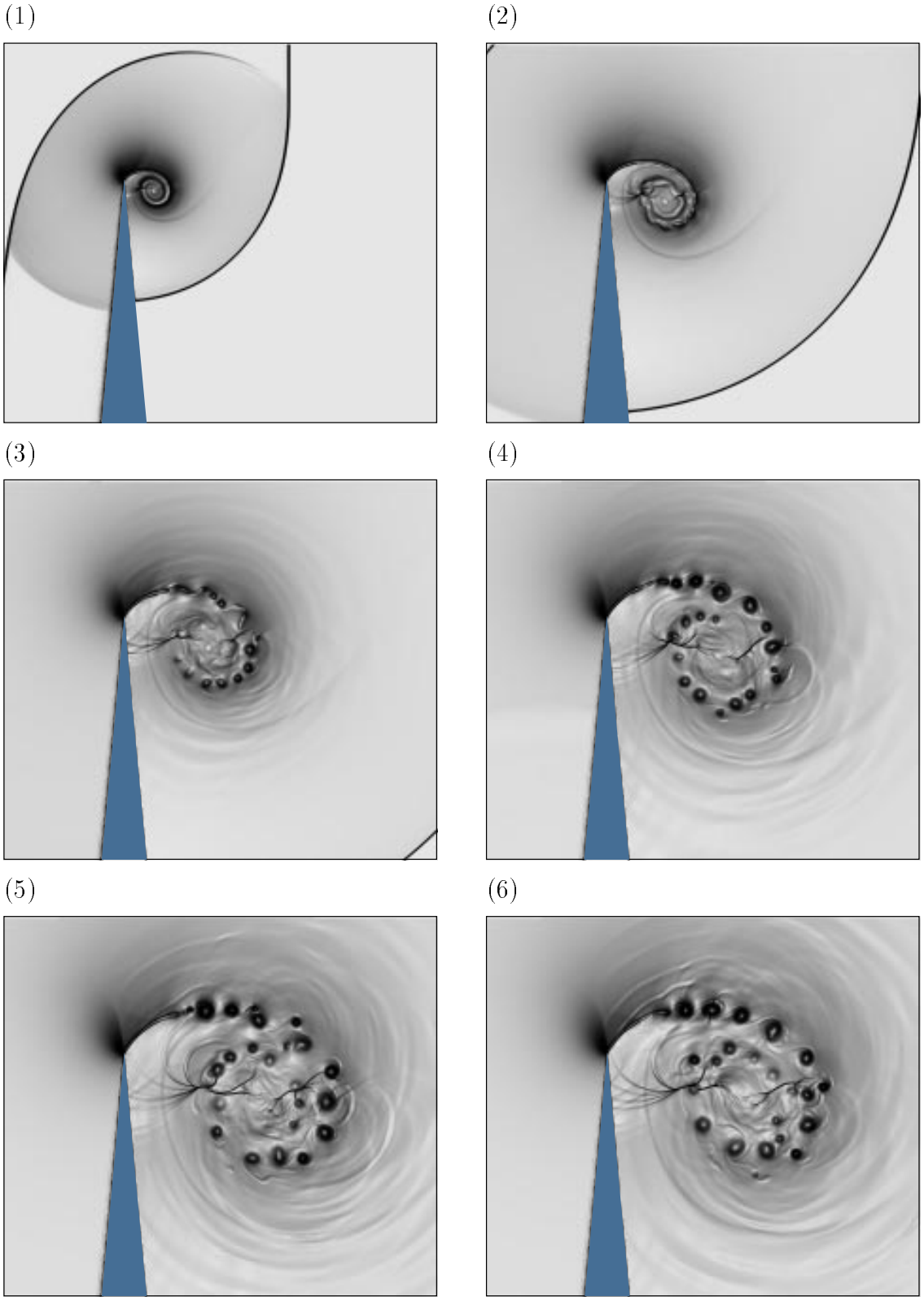


Figure 13: Evolution of a vortex sheet due to a shock wave diffracting over a knife edge.

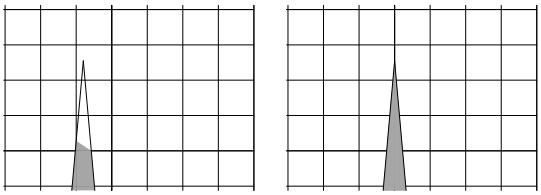


Figure 14: The tip of the knife edge with and without blunting.

credence to the minutiae at late times in the simulation since these are controlled by vestigial numerical diffusion and will thus vary from scheme to scheme. Indeed, for the vortex produced by a $M_S = 1.5$ shock wave diffracting around a 90^o corner, the variations in structure with changes in numerical scheme are far greater than the changes here due to blunting[31].

5 CONCLUSIONS

While a Cartesian boundary-cum-mesh refinement approach can undoubtedly produce spectacular results, it must be realized that there is no concise recipe for success. Consequently, we feel that the high development costs will continue to act as a deterrent and so limit the popularity of this approach. Nevertheless, if maximum resolution is sought, the advantages of the present scheme far outweigh its development costs. Moreover, since the basic machinery is not tied to any one integration scheme and it forms a reliable framework that can be readily exploited by a variety of applications, the effective costs are to some extent diminished. As they are every time the method is used, simply because there are no longer any grid generation costs to worry about.

It is also worth noting that a Cartesian boundary scheme becomes more efficient as the resolution of the computation increases, because the cut-cells occupy an increasingly smaller volume in space and therefore introduce less of an overhead. Moreover, a Cartesian scheme does not distort the mesh in sensitive parts of the flow field, as sometimes happens with body-fitted grids to the detriment of the computed solution.

Finally, despite its logical complexity we have demonstrated that the present scheme can exploit parallel computing engines efficiently and so it is not likely to be overtaken by advances in computer architectures which would make it redundant.

Acknowledgements

I would like to thank Drs. S. Karni, K.G. Powell and V.Venkatakrishnan for their comments which helped improve this paper. This paper will appear in the proceedings volumes of ECCOMAS 94, published by John Wiley and Sons Limited.

References

[1] R. Löhner, *Adaptive H-refinement on 3-D unstructured grids for transient problems.* AIAA Paper 89-0653 (1989).

[2] J.L. Steger and J.A. Benek, *On the use of composite grid schemes in computational aerodynamics.* Computer Meth. Appl. Mech. Eng., **64**(1987), pp. 301–320.

[3] J.F. Thompson and N.P. Weatherill, *Aspects of numerical grid generation: current science and art.* AIAA Paper 93-3539-CP (1993).

[4] P.R. Woodward, Proc. Nato workshop in Astrophysical Radiation Hydrodynamics, Munich, Germany. Nov. 1983.

[5] S.A. Bayyuk, K.G. Powell and B. van Leer, *A simulation technique for 2-D unsteady inviscid flows around arbitrarily moving and deforming bodies of arbitrary geometry.* AIAA Paper 93-3391-CP (1993).

[6] M.J. Berger and R.J. LeVeque, *An adaptive Cartesian mesh algorithm for the Euler equations in arbitrary geometries.* AIAA Paper 89-1930-CP (1989).

[7] Y-L. Chiang, B.van Leer and K.G. Powell, *Simulation of unsteady inviscid flow on an adaptively refined Cartesian grid.* AIAA Paper 92-0443 (1992).

[8] D.K. Clarke, M.D. Salas and H.A. Hassan, *Euler calculations for multielement airfoils using Cartesian grids.* AIAA Journal, Vol. **24**(1987), No. 3, pp. 353–358.

[9] W.J. Coirier and K.G. Powell, *An accuracy assessment of Cartesian-mesh approaches for the Euler equations.* AIAA Paper 93-3335-CP (1993).

[10] S.A.E.G. Falle and J. Giddings, *Body capturing using adaptive Cartesian grids.* Numerical Methods in Fluid Dynamics 4, Clarendon press, Oxford (1993), pp. 337–343.

[11] A.D. French, *Solutions of the Euler equations on Cartesian grids.* Ph.D. thesis, College of Aeronautics, Cranfield Institute of technology (1991).

[12] R.L. Gaffney, H.A. Hassan and M.D. Salas, *Euler calculations for wings using Cartesian grids.* AIAA Paper 87-0356-CP (1987).

[13] B.P. Gerasimov and S.A. Semushin, *An Eulerian method for calculation of gas motion in a Varying Region.* Lecture Notes in Physics, Vol. **170**(1982), p. 211.

[14] J.E. Melton, F.Y. Enomoto and M.J. Berger, *3D automatic Cartesian grid generation for Euler flows.* AIAA Paper 93-3386 (1993).

[15] R.B Pember, J.B. Bell, P. Colella, W.Y. Crutchfield and M.L. Welcome, *Adaptive Cartesian grid methods for representing geometry in inviscid compressible Flow.* AIAA Paper 93-3385-CP (1993).

[16] A. Priestley, *Roe's scheme, Euler equations, Cartesian grids, non-Cartesian geometries, rigid walls and all that.* Univ. Reading, Dep. Math., Num. Anal. Rep. 14/87, (1987).

[17] J.J. Quirk, *An alternative to unstructured grids for computing gas dynamic flows around arbitrarily complex two-dimensional bodies.* Computers & Fluids, Vol. **23**(1994), pp 125–142.

[18] D.P. Young, R.G. Melvin, M.B. Bieterman, F.T. Johnson, S.S. Samant and J.E. Bussoletti, *A locally refined rectangular grid finite-element method: application to computational fluid dynamics and computational physics.* J. Comput. Phys., **62**(1991), pp. 1–66.

[19] D.De Zeeuw and K.G. Powell, *An adaptively refined Cartesian mesh solver for the Euler equations.* J. Comput. Phys., **104**(1993), pp. 56–68.

[20] J.J. Quirk, *An adaptive grid algorithm for computational shock hydrodynamics.* Ph.D. thesis, College of Aeronautics, Cranfield Institute of technology (1991).

[21] J.R. Rice, *Numerical computation with general two dimensional domains.* ACM Trans. Math. Software, Vol. **10**(1984), pp 443–452.

[22] J.R. Rice, Private Communication, August 1992.

[23] J.D. Foley, A.van Dam, S.K. Feiner and J.F. Hughes, *Computer graphics, principles and practice.* 2nd edn. Addison Wesley (1990), p.555.

[24] J.J. Quirk, AGARD mission #2C20622, December 1992.

[25] M.J. Berger and P. Colella, *Local adaptive mesh refinement for shock hydrodynamics.* J. Comput. Phys., **82**(1989), pp. 67–84.

[26] J.J. Quirk and U.R. Hanebutte, *A parallel adaptive mesh refinement algorithm.* NASA CR-191530, ICASE Report No. 93-63, (1993).

[27] J.J. Quirk, *Dynamic load balancing strategies for parallel adaptive mesh refinement.* In preparation.

[28] R.L. Graham, *Bounds on certain multiprocessing anomalies.* SIAM J. Appl. Math., **17**, No. 2 (1969), pp 416–429.

[29] B. Sturtevant and V.A. Kulkarny, *The focusing of weak shock waves.* J. Fluid Mech., **73** part 4 (1976), pp. 651–671.

[30] M. van Dyke, *An album of fluid motion.* The Parabolic Press, Stanford, California, 1992, p. 49 and p. 145.

[31] Shock Waves, An International Journal, **1** No.4 (1991).