NASA Contractor Report 189612
ICASE Report No. 92–7

ICASE

AN ALTERNATIVE TO UNSTRUCTURED GRIDS FOR COMPUTING GAS DYNAMIC FLOWS AROUND ARBITRARILY COMPLEX TWO-DIMENSIONAL BODIES

James J. Quirk

Contract No. NAS1–18605
February 1992

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, Virginia 23665–5225

Operated by the Universities Space Research Association

NASA
National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

An Alternative to Unstructured Grids for Computing Gas Dynamic Flows Around Arbitrarily Complex Two-Dimensional Bodies

James J Quirk [1]
Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA 23665, USA.

ABSTRACT

Within the shock-capturing community, the need to simulate flows around geometrically complex bodies has resulted in an inexorable shift away from schemes which employ body-fitted grids to schemes which employ unstructured grids. Although unstructured grids are undeniably effective, in view of the increasing reliance placed on computational results, such a wholesale shift in mentality should give cause for concern. The concept of using several computer codes to cross check numerical results becomes ill-founded if all codes follow the same methodology. In this paper we describe an alternative approach for dealing with arbitrarily complex, two-dimensional geometries, the so-called cartesian boundary method.

Conceptually, the cartesian boundary method is quite simple. Solid bodies blank out areas of a background, cartesian mesh, and the resultant cut cells are singled out for special attention. However, there are several obstacles that must be overcome in order to achieve a practical scheme. We present a general strategy that overcomes these obstacles, together with some details of our successful conversion of an adaptive mesh algorithm from a body-fitted code to a cartesian boundary code.

[1]This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-18605 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23665.

1 Introduction

Given calculations such as Löhner's[13] simulation of a blast wave impinging on a tank, there can be no doubt within the shock capturing community that unstructured grids provide a highly effective means of simulating flows around geometrically complex bodies. Indeed, such is the success of unstructured grids, there is a danger that they will become *de rigueur*. This would be a cause for some concern given the increasing reliance placed on computational results. If nothing else, the concept of using several codes to cross check numerical results becomes ill-founded if all codes follow the same methodology. Besides, the superiority of unstructured grids is not clear-cut. For example, there is numerical evidence to suggest that for very strong shocks, schemes which employ unstructured grids suffer from larger phase errors than do schemes which employ structured grids[18].

In this paper we describe an alternative approach for dealing with complex, two-dimensional geometries, the so-called cartesian boundary method. Conceptually, this method is quite simple. Solid boundaries blank out areas of a background cartesian mesh, and the resultant cut cells receive special attention during the integration of the flow solution. However, this simplicity of concept belies the obstacles that must be overcome in order to achieve a practical scheme. Whilst these obstacles are far from insurmountable they are often perceived as stumbling blocks, hence the dearth of schemes which employ the cartesian boundary method.

Now it is not our intention to rubbish unstructured grid methods. This would be foolish, for it is the specific problem in hand that ultimately determines which solution strategy is best. Instead, we wish to demonstrate that contrary to popular opinion, a structured grid scheme can match the geometric flexibility exhibited by unstructured schemes. Accordingly, we do not extol the advantages of structured grids over their unstructured counterparts. Nor do we survey the few cartesian boundary schemes that appear in the literature, for most of these schemes have only been shown to work for stylized geometries. Suffice it to say, the majority of schemes have been developed for steady state, transonic flow calculations, examples being[5, 6, 14]. To the best of our knowledge, only Berger and Le Veque[3], and Chiang *et al*[7] have tackled unsteady flows which involve strong shock waves. Where appropriate, differences between these two schemes and our method are discussed in the next section and so need not be considered here.

It has been our experience that the major obstacle to developing a cartesian boundary scheme lies in formulating a general strategy that can cope with truly com-

plex geometries. Consequently, in this paper we have concentrated on describing the practical formulation of our method rather than on presenting its theoretical justification. To this end, in section 2 we present an overview of the scheme highlighting some of the problems that needed to be overcome. This is followed by detailed descriptions for each major constituent of the scheme. In order to produce an algorithm that is competitive compared with unstructured grid methods, it is necessary to combine the cartesian boundary scheme with some form of local mesh refinement. In section 4 we give details of one suitable mesh refinement scheme, the Adaptive Mesh Refinement (AMR) algorithm. Then, so as to demonstrate the effectiveness of our algorithm for simulating shock hydrodynamic flows around complex geometries, a section of results is presented. Finally, in section 6 we give the main conclusions that we have drawn from this work.

2 Overview

In essence, our cartesian boundary scheme follows a finite-volume approach. Thus it makes no difference to the mechanics of the scheme as to which equations are integrated, they simply need to be given in conservation form. However, we do assume that the computational grid contains a cell-centred projection of the flow solution, for solid wall boundary conditions are applied via a local reflection at the wall. As French[9] has demonstrated, it is possible to develop a cartesian boundary method for cell-vertex schemes but, because of the number of different situations that can arise, this necessarily results in an algorithm which is more unwieldy than its cell-centred counterpart.

Compared to other types of grid, cartesian boundary grids appear straightforward. Solid bodies merely blank out areas of a background, cartesian mesh. This gives rise to three classes of mesh cell. Namely, cut cells which lie along the surface of a body, solid cells which lie wholly within a body, and uncut cells which lie outside a body. But a cut cell may be further categorized as one of several types depending on the number and relative positions of the intersections of the body with the cell. So given an arbitrary set of bodies, the process of determining the exact nature of each cell in the mesh is not trivial. Moreover, since our simulations employ an adaptive grid this cell-type information cannot be gathered as a one-off at the start of each calculation; it must be gathered each time the grid is adapted. Therefore considering that a typical calculation involves several hundred grid adaptions, it is imperative that the gathering process be efficient.

Our gathering process starts by tracing the outline of each body so as to find all the intersections between the grid and the specified input geometry. These intersections are then collated to find the types and locations of all the cut cells. Given this information, it is then a simple matter to scan the mesh, thereby determining which cells are solid, and which cells are uncut. Since only cut cells are examined in detail this method proves fairly efficient, but a few insidious problems must be overcome in order that the method be made foolproof. For example, suppose that the input geometry consists of a closed body formed from several straight line segments. A naïve algorithm which employed floating point arithmetic to calculate the intersection points could well fail if one of the line junctions lay on a grid line. Because there is no control over rounding errors, the end of one line segment could effectively lie on one side of a grid interface, with the start of the next line segment lying on the other side. Thus an intersection would not be registered. Although infrequent, such problems thwarted our early attempts at coding a general purpose algorithm for gathering the cut cell information. Our latest method avoids such problems by finding intersections relative to a lattice of finite resolution.

Once the cell-type information has been gathered, a finite-volume scheme may be used to integrate the discretized flow solution. We use the two step method proposed by Hancock[11]. First, a form of MUSCL interpolation[17] is used to reconstruct the flow solution within each mesh cell. An intermediate solution is then found by advancing this reconstructed solution by half a time step. This intermediate solution defines a set of left- and right-hand states for a series of Riemann problems. The solutions to these Riemann problems provide a set of upwinded interface fluxes which are used to integrate the original flow solution forward by one full time step. Since we are using a cell-centred projection the calculation of flux integrals for cut cells presents no special difficulties. As is common practice, the flux for an interface which forms part of a solid surface is found by computing a reflected Riemann problem. However, as it stands, our code reduces to first-order along a solid boundary because the reconstruction of the flow solution within a cut cell is zeroth-order. So far this has not proved to be a limitation, for accuracy can always be improved via the use of local mesh refinement. But, if needs be, matters could be improved by adopting the cut cell reconstruction technique proposed by De Zeeuw and Powell[6].

Since cartesian boundary grids result in some very small cut cells, there is one important step that must be added to the above procedure in order to ensure the stability of the scheme. In essence, stability problems are circumvented by absorbing

small cells into large cells. As will be described in section 3.5, a set of lists is produced which link small cells to large, neighbouring cells. In this context, if the area of a cell is less than half that of an uncut cell then it is deemed to be small, otherwise it is deemed to be large. Note that a single list may contain more than two cells, but no cell appears in more than one list. Once a residual for each mesh cell has been calculated via a surface flux integral, the residuals for the cells contained by a given list are replaced by their volume weighted average. This new residual is equivalent to that which would have been found if a separate flux integral had been performed for the single cell formed from the union of all the cells in the list. Therefore this procedure is just a convenient way by which to compute the residual for some odd shaped cell. It should not be construed as a smoothing process. Finally, these modified residuals are used to update the flow solution to the next time level. Thus, provided the CFL condition is satisfied for cells as small as half an uncut cell, the integration process is stable. A similar cell absorption technique was used by Clarke *et al*[5] to stabilise their steady-state, transonic airfoil calculations. However, as described, it is doubtful if their method would work for bodies which contain re-entrant corners.

2.1 Comment

In all probability, the unpopularity of cartesian boundary schemes is due to the perceived difficulties associated with overcoming the stability problems brought about by disparate cell sizes, but to a large extent any fears are unfounded. Indeed, given a finite-volume mentality, our strategy follows almost trivially. Moreover, De Zeeuw and Powell[6] have shown that for steady-state calculations a local time stepping strategy is sufficient to ensure stability, but such strategies are already commonplace as a means to accelerate the convergence rate. On the other hand, Berger and Le Veque[3] adopted a much more ambitious strategy for ensuring the stability of their cartesian boundary scheme. In essence, they employ a large time step generalisation of Godunov's method. That is, following the solution to a set of Riemann problems, the flow solution is evolved by tracking individual waves across the mesh. As a wave crosses a cell, either partially or wholly, so the flow variables are adjusted accordingly. Thus, although the method is explicit, in principle, it does not suffer a stability restriction on the size of the time step that may be used to evolve the flow solution. Their motivation for using this approach is to avoid the loss in resolution associated with absorbing small cells into larger cells. However, given the complexity of their method, we feel that the ends do not justify the means.

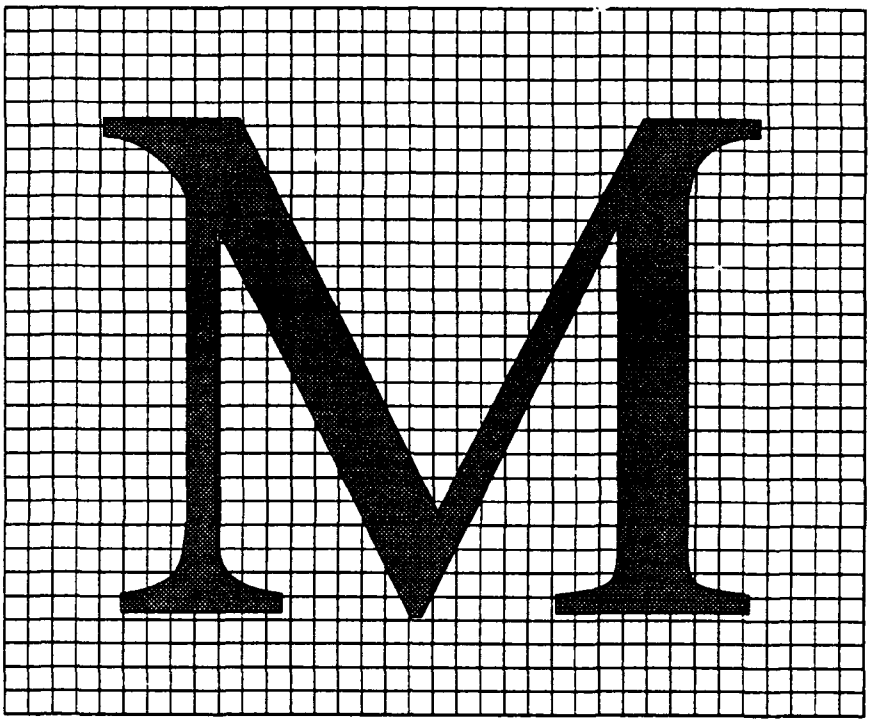
Firstly, if a cell is genuinely small, it matters not one jot if it is absorbed into a neighbouring cell. Secondly, the small loss in resolution associated with absorbing cells which are close to half the size of an uncut cell can be more than made up via local mesh refinement. Besides, their large time step method is not foolproof; witness the fact that for very small cells they report the need to use some form of cell absorption procedure. It is not hard to envisage why the large time step method is occasionally found wanting. For example, no account is taken of the wave interactions that might occur during the course of a single time step. Now given that waves reflect from solid surfaces, interactions are bound to occur in the neighbourhood of uncut cells. For strong waves any interactions will be highly nonlinear. Hence, failure to allow for the interactions will lead to erratic results. Even when only weak waves are involved, in which case it is relatively safe to ignore any interactions, problems arise in the vicinity of re-entrant corners. For it becomes difficult, logistically, to keep track of the area swept out by a single wave, given that it may well rebound several times between the surfaces that form the corner.

3 Algorithm Details

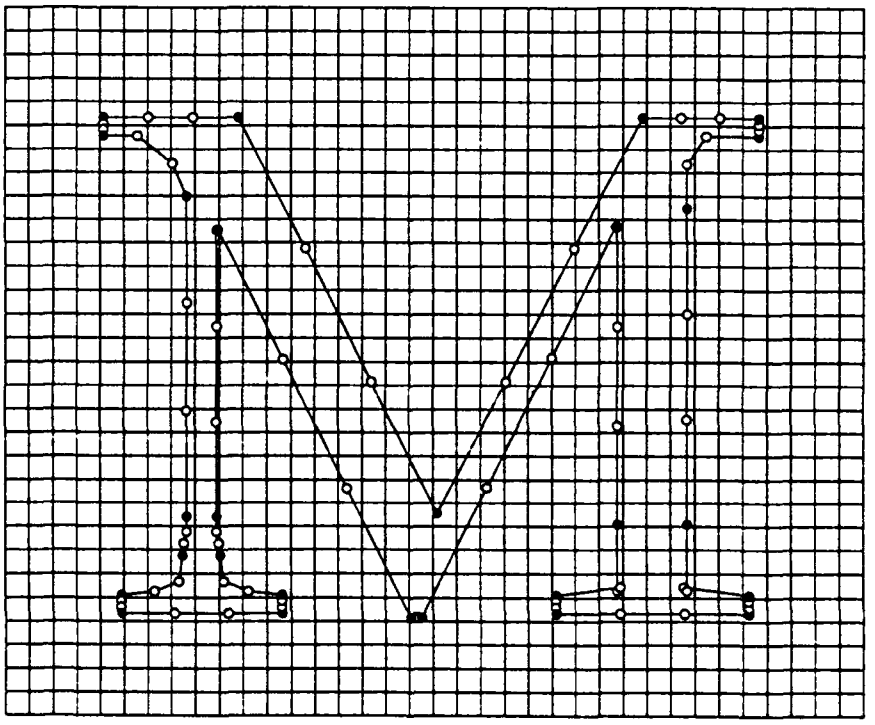
As mentioned previously, the major obstacle to developing a cartesian boundary scheme lies with formulating a general algorithm that can handle truly complex geometries. Now, a general algorithm necessarily contains many mundane components, so it would be inappropriate for us to describe our scheme in complete detail. Instead, we simply detail its main elements.

3.1 Input Geometry

If a cartesian boundary scheme is to be genuinely useful, it must be able to cope with a wide range of input geometry without any user intervention whatsoever. In our scheme the geometry is specified via an arbitrary number of cubic-Bézier curves which provide the outlines for one or more solid bodies. Only two minor restrictions apply: first, no two bodies can overlap one another; second, each outline must form a simple closed curve. But since it makes no sense to transgress these restrictions, to all intents and purposes, the scheme can cope with arbitrary shaped bodies. For example, the letter M could be input using the 29 Bézier curves shown in figure 1. Note that each Bézier curve consists of 4 control points. Two control points fix the endpoints of the curve, and two control points fix the slope of the curve at



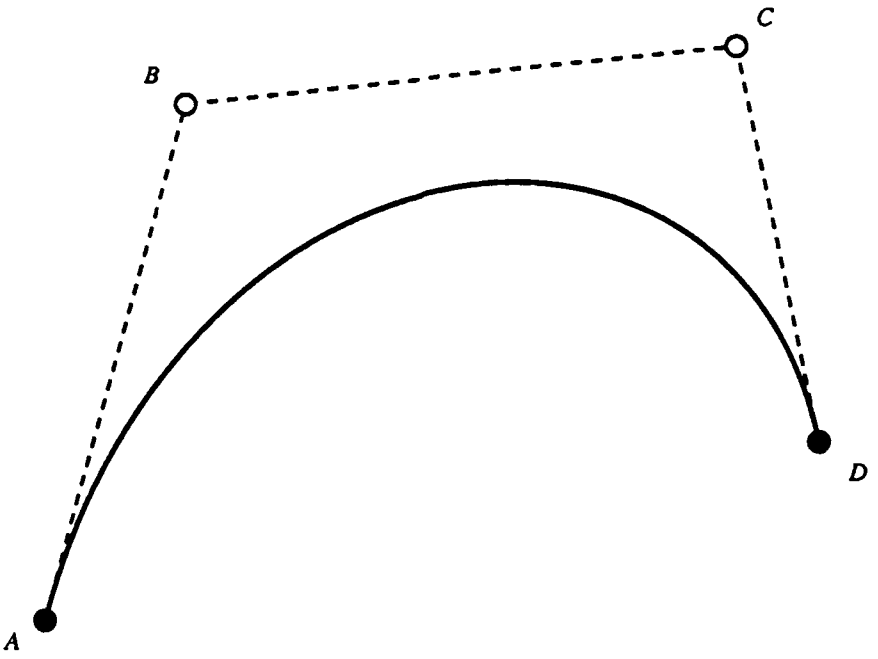
(a)



(b)

Figure 1: Example of input geometry. (a) Required shape. (b) Input Bézier curves. (c) Formulation of a cubic-Bézier curve.

(c)



For the control points $\{A, B, C, D\}$ the resultant cubic-Bézier curve may be written in the form,

$$\begin{aligned} X(t) &= B_0 X_A + B_1 X_B + B_2 X_C + B_3 X_D, \\ Y(t) &= B_0 Y_A + B_1 Y_B + B_2 Y_C + B_3 Y_D. \end{aligned}$$

Where,

$$\begin{aligned} B_0 &= (1-t)^3, \\ B_1 &= 3t\ (1-t)^2, \\ B_2 &= 3t^2(1-t), \\ B_3 &= t^3, \end{aligned}$$

and t is limited to values between 0 and 1.

Figure 1: For caption, see facing page.

these endpoints. These two types of control point are marked by solid circles and open circles respectively. The choice to use Bézier curves over some other form of parametric representation was made solely on our knowledge of computer graphics. In which field this type of curve is widely used as a cheap, yet flexible means of specifying geometrical information. Indeed, many useful algorithms pertaining to Bézier curves are available from standard texts on computer graphics[8, 10]. However, since all curves are effectively reduced to a series of straight line segments it would be a simple matter to change to some other parametric representation.

3.2 Locating Grid Intersections

Given some input geometry, the next step is to find all the intersection points between the cartesian mesh and the outlines of the solid surfaces. As was described in section 2, this task is not as innocuous as it appears. If the cartesian boundary scheme is to be robust, it is essential that control be exercised over floating point round-off errors. In essence, we eliminate round-off errors by finding the intersection points relative to some lattice of finite resolution. Thus we can guarantee that every cut cell has at least two intersection points. The importance of this fact will become clear later on.

Briefly, the location process works as follows. The outline of each body is traced in an anti-clockwise direction, Bézier curve by Bèzier curve, individual curves being stroked as a series of straight line segments. Now algorithms exist which minimize the number of line segments required to draw a Bézier curve to some prescribed accuracy[8, 10]. Therefore it does not take an inordinate number of segments to get a good representation of the outline. As a straight line is stroked, so the intersection points, if any, between the grid and the line are found and saved in a list. Thus a complete list is generated of the intersection points between the mesh and the input outlines. The only step of the location process which merits a detailed description is that which strokes a single straight line segment.

Just prior to being stroked, a line segment is clipped against the mesh. So, the algorithm for determining the grid intersection points need only work for the case where the line lies wholly within the mesh. Therefore it can assume that the endpoints of a line are available in the following form. An (I, J) co-ordinate pair identifies a specific mesh cell, and an (i, j) offset pair identifies a specific location within that cell. So as to do away with the need for floating point operations within the stroking algorithm, each cell is effectively split into a matrix of N by N pixels. An (i, j) offset simply identifies one of these discrete elements. Therefore, $0 \leq i, j \leq (N - 1)$. Note

that to ensure accuracy, N should be large; in our code, N is set to 2^{20}. For brevity we denote the discrete location of an endpoint by $<I, J: i, j>$. Although rounding errors inevitably occur when finding the discrete location for a point, (x, y), specified in world co-ordinates, they do so in a controlled manner. Therefore given a series of line segments: $(x_1, y_1) \rightarrow (x_2, y_2), (x_2, y_2) \rightarrow (x_3, y_3)$, *etc.*; the discrete location for the start of one line can be guaranteed to match that for the end of the previous line in the series.

The inner workings of the stroking algorithm are best explained by considering a specific example. The grid intersection points for the line shown in figure 2 are found using the algorithm given in figure 3. First, a simple check is applied to see whether or not the two endpoints lie within the same mesh cell. If this is the case, the line cannot intersect the grid so no further processing need be done. Otherwise, one or more intersections need to be found. The algorithm starts at the left-hand end of the line and proceeds to the right by way of a series of jumps: $a, b, c, \ldots, h$. With each jump a new intersection, j^n, is found between the line and a vertical grid line. This intersection is just an offset like that used to specify the endpoints of the line. Note that if the size of this intersection offset is larger than $(N - 1)$, the line must just have crossed a horizontal grid line[2]. In which case: the horizontal intersection offset, i, is found by interpolation; j^n is reduced by N to give the correct vertical offset, j; the J co-ordinate is incremented by one so as to move up a row. As an intersection is encountered, so it is saved in a list and the two adjoining mesh cells are marked as being cut. Note the black dots shown in figure 2 correspond to the location (I, J) just before each jump is taken.

Although the basic concept behind the stroking algorithm is straightforward, several subtleties exist. The two most important ones are as follows. First, when calculating the increment added to j, given an increment to i, it is necessary to keep track of fractions of a pixel. Whenever the running total for these fractions accrue one unit, the increment to j is increased accordingly. Therefore, with reference to figure 3, given a cumulative change in i of ΔI, the sum of the increments to j add up to ΔJ, exactly[3]. Thus the stroking algorithm cannot fall short of the end of a line, thereby missing an intersection point. Nor can it march past the end of a line, thereby generating a spurious intersection point. Second, care should be taken to ensure that the evaluations of N_j and N'_j do not cause an arithmetic overflow. Note

[2]Effectively, an intersection is registered whenever an offset clicks over from $(N - 1)$ to N.

[3]ΔI and ΔJ correspond to the width and height of the line, respectively, in terms of pixels.

that normal 32 bit integer arithmetic is inadequate for our purposes; the largest intermediate value that could possibly occur during the evaluation of N_j is $N^2(I_2 - I_1)$, with N set to 2^{20} this value is bound to exceed 2^{32}. Fortunately, a simple ruse may be executed using double precision variables. Although we are only interested in integer arithmetic, if $N, N_j, \Delta J$ and ΔI are declared to be double precision, we can take advantage of the 52 bit mantissa offered by such variables which adhere to the IEEE standard. Therefore, with N set to 2^{20}, the integers N_j and N'_j will be evaluated exactly for values of $(I_2 - I_1)$ as large as 4096.

Finally, while the algorithm given in figure 3 only works for lines whose slope lie between 0° and 45°, analogous algorithms can be formulated for each of the seven remaining octants. Therefore it is relatively straightforward to formulate a combined algorithm which can cope with lines of arbitrary slope.

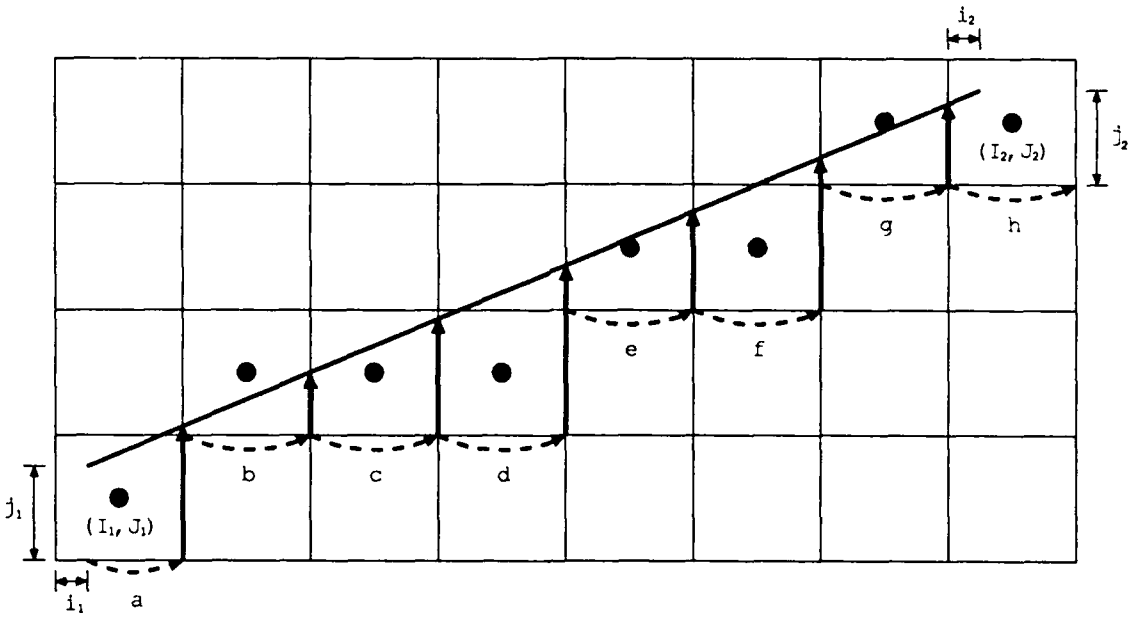


Figure 2: Example of the line stroking procedure.

3.3 Cut Cell Information

Once all the grid intersection points have been located, a simple collating procedure is used to determine the cut cell information. Now we allow just three basic types of cut cell, which together with four different orientations gives rise to the 12 types of cut cell shown in figure 4. Note that each of these cut cells has only two intersection points.

```
Procedure Stroke_Line(<I1, J1 : i1, j1>,<I2, J2 : i2, j2>)
  if (I1 ≠ I2 ∪ J1 ≠ J2) {                % Does the line lie within a single mesh cell?
    <I,J : i, j> = <I1,J1 : i1, j1>       % No, so get start of line and continue.
    ΔI = (I2 − I1) *N + (i2 − i1)         % Find width of line's bounding box.
    ΔJ = (J2 − J1)*N + (j2 − j1)          % Find height of line's bounding box.
    j'  = j° = 0                          % Reset remainder running totals.
    while(true) {
      Nj = ((N − i) * ΔJ)/ΔI              % Find increment to j given a step of (N-i).
      N'j =  (N − i) * ΔJ − Nj * ΔI       % Calculate remainder of j increment.
      jn  = j + Nj                        % Find new intersection point.
      j'  = j'+ N'j                       % Compute Running total for remainders.
      if ( j' ≥ ΔI) {                     % Has the running total reached one unit?
        jn = jn + 1                       % Yes, so bump intersection point by one.
        j'  = j'  − ΔI                    % And adjust running total accordingly.
      }
      if (jn ≥ N) {                       % Has a horizontal intersection occurred?
        i = i + ((N − j) * ΔI − j°)/ΔJ    % Yes, so find the intersection point.
        if (I = I2 ∩ i > i2) break        % Exit if end of line is exceeded.
        Ptr = Save_Intersection(i)        % Save intersection and mark cut cells.
        Save_Northern_Edge(Ptr, I, J)
        Save_Southern_Edge(Ptr, I, J + 1)
        jn= jn−N                          % Adjust jn to range [0,N-1].
        J = J +1                          % Move up a cell.
      }
      j = jn                              % Replace the old intersection.
      j°= j'                              % Save for next horizontal interpolation.
      if (I ≥ I2) break                   % Exit if end of line is exceeded.
      Ptr = Save_Intersection(j)          % Save vertical intersection.
      Save_Eastern_Edge(Ptr, I, J)        % And mark cut cells.
      Save_Western_Edge(Ptr, I + 1, J)
      I = I + 1                           % Move to next vertical grid line.
      i  = 0
    }                                     % Repeat.
  }
End Procedure
```

Figure 3: Procedure to stroke a straight line; limited to slopes between 0° and 45°.

But, there is no upper limit to the number of intersections that might conceivably be found for any one mesh cell. So, the type of each cell is determined from its first and last intersection points. Remember the intersection points were found by tracing out the boundary, and so the intersection points for any one cell are listed in strict order. Therefore it is not unreasonable to approximate the solid boundary by a line joining the first and last intersection points. Under normal circumstances, a cell having more than two intersection points merely indicates that the mesh is too coarse to resolve the geometry properly.

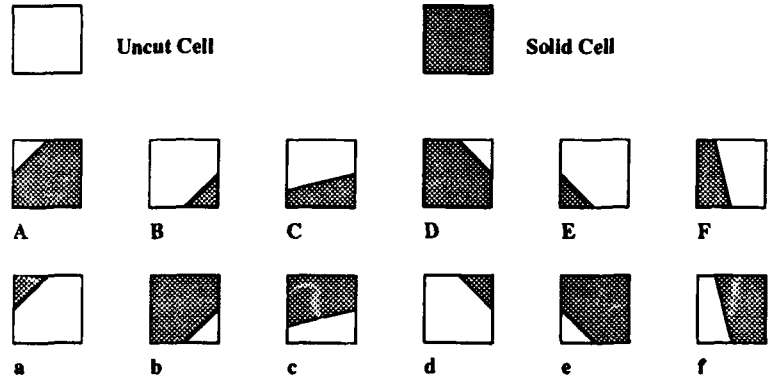
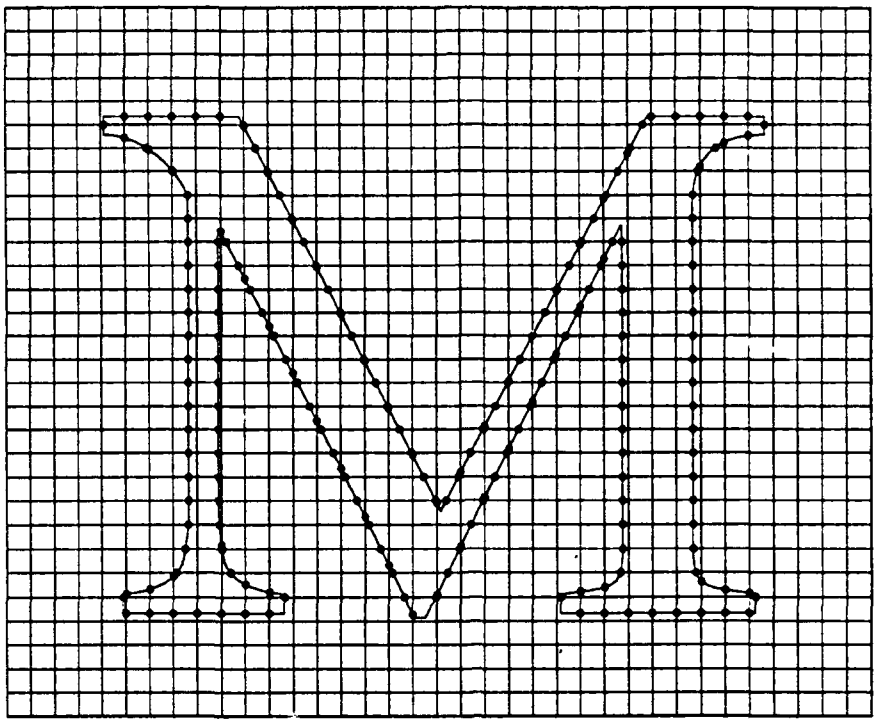


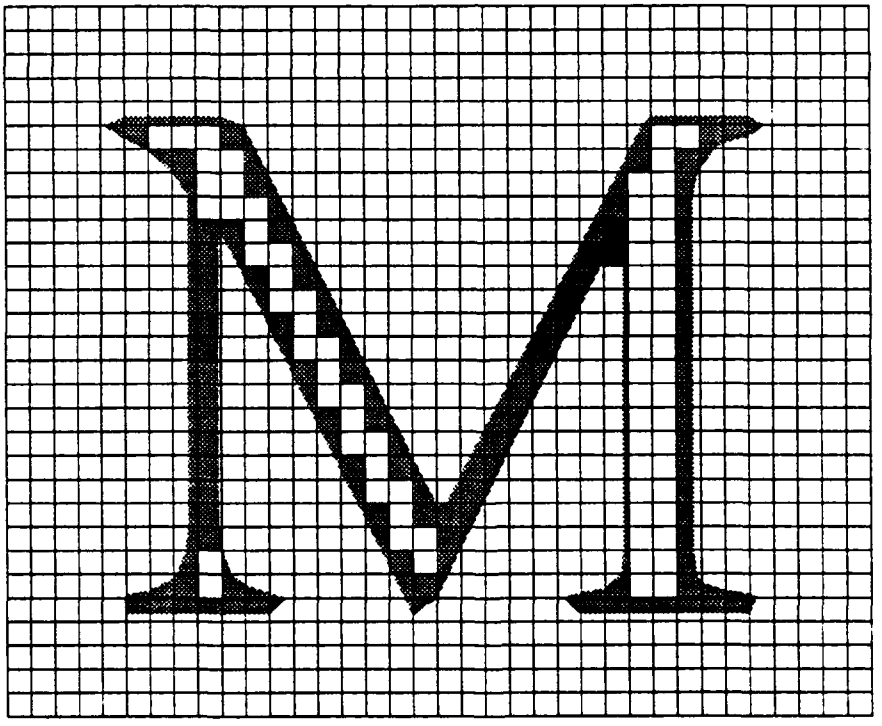
Figure 4: Different types of cell.

Given these two intersections, it is a trivial matter to determine the type of the cut cell. For example, suppose the first intersection lies along the western edge, and the second intersection lies along the northern edge. In which case, the cell must be of type *a*. Note since outlines are traced in an anti-clockwise direction the cell cannot be of type *A*. It should now be clear why it is important to ensure that at least two intersections are found for each cut cell. For if rounding errors resulted in there being only one intersection point it would be impossible to determine the type of the cell.

Figure 5 shows the grid intersection points for the letter M, and the corresponding cut cells produced by the intersection collation procedure. Note that sharp corners are inevitably blunted. While such blunting could largely be avoided by the introduction of further cell-types this would necessarily lead to a more complicated scheme. We simply circumvent the problem of blunting by using local mesh refinement; the finer the mesh, the smaller the degree of blunting. Also, note that the collation procedure has flagged some cells as being degenerate, such cells are marked as dark squares. Basically, a cell is flagged as being degenerate whenever it is too coarse to provide an unambiguous, local representation of an outline. Degenerate cells must be refined.



(a)



(b)

Figure 5: (a) Grid Intersection points for the letter M. (b) Cut cells resulting from the intersection collation procedure.

3.4 Finding Solid Cells

Once all the cut cells have been located it is a simple matter to scan the mesh thereby finding all the solid cells. This scanning procedure involves four sweeps. First, each horizontal strip of cells is scanned from left to right. During which, if a cell is found to be uncut then it is flagged as being solid depending on whether a switch is on or off. At the start of each row this switch is set to off, that is uncut cells are not to be flagged as solid. Scanning one of the cell-types $\{A, e, f\}$ turns the switch on, while scanning one of the types $\{a, E, F\}$ turns it off. This simple procedure would suffice if only convex bodies were allowed. But, to provide a foolproof method of locating all the solid cells contained by an arbitrary body it is necessary to perform a further three sweeps which scan the mesh from right to left, top to bottom, and bottom to top. Note that each sweep uses a different set of cells to toggle the flagging switch.

3.5 Combination Cells

After all the cut cells have been located, the following procedure determines which cells should be grouped together so as to ensure the stability of the flow integration process. In essence it produces a set of lists, each list identifies the members for one distinct group of cells.

The following procedure is applied to each cut cell in turn. First, the cell is checked to see if it is small. If the cell is larger than half that of an uncut cell it does not need to be linked to another cell, so no further processing need be done. Otherwise, an offset is found which points to a prospective combination cell. This combination cell lies adjacent to the cut cell, and is the cell that would be entered when leaving the cut cell along a normal that starts from the midpoint of the edge that marks the solid boundary. Note no trigonometry is required to find this offset. For example, for a type-A cell the offset can only be $(0, 1)$ or $(-1, 0)$. The choice of offset is readily determined from the relative positions of the two intersections which fix the cell. Further processing depends on whether the cut cell or the prospective combination cell already belong to a list or not. If neither cell is in a list, they are both added to a new list. If just the combination cell is part of a list, the cut cell is added to the existing list. Similarly, if the cut cell is part of a list, the combination cell is added to the existing list. Now if both cells are already in a list, one of two situations is possible. First, both cells are already part of the same list, in which case nothing need be done. Second, the cells belong to different lists, in which case one list is destroyed it's contents being first added to the other list. This ensures that a

cell does not belong to more than one list.

One complication must be added to the above procedure in order that concave corners are handled properly. For example, consider a type-A cell and suppose that the offset of the prospective combination cell was found to be $(-1, 0)$. If this combination cell is of type D, the two cells form a sharp concave corner. So the offset is taken to be $(0, 1)$. Similarly, if the offset was found to be $(0, 1)$ and the combination cell was of type e, the offset is replaced by $(-1, 0)$. The choice of combination cells for the cut cells $\{D, b, e\}$ is handled analogously. Figure 6 shows the grouping of the combination cells in the vicinity of the central concave corner for the body shown in figure 1. Note, so as to distinguish between groups which adjoin one another, each combination group is coloured using one of three shades. No significance should be attached to the choice of shade for a specific combination group.

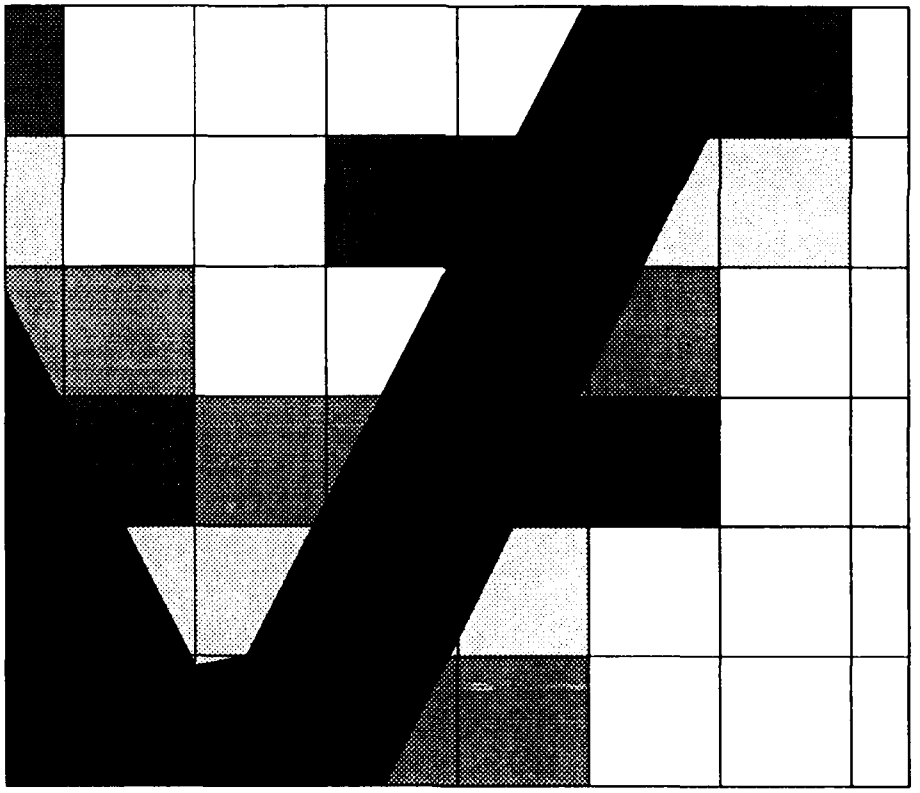


Figure 6: Combination groups produced in the vicinity of a sharp concave corner.

3.6 Comment

It should now be apparent that the mentality required to develop a cartesian boundary scheme is somewhat different to that normally used for computational fluid dynamics, hence the level of detail in this paper. Indeed, parts of our method parallel certain basic algorithms from the field of computer graphics. For example, the procedure given in figure 3 is similar in spirit to Bresenham's classic line drawing algorithm. Given these parallels, it seems likely that our method could be improved upon via the adaptation of more sophisticated graphics algorithms. Nevertheless, as it stands the method is efficient, for only cut cells are ever examined in detail. Thus, the combined workload for the procedures described in this section grows roughly with the square root of the number of cells contained by the cartesian mesh, as against the effort required to integrate the flow solution which grows directly with the number of mesh cells. Moreover, for the calculations presented in section 5, in terms of the overall workload, less than 3% of the computational effort was expended on these procedures, the rest being expended on the AMR algorithm. Therefore, any improvements that could be made would prove largely inconsequential.

4 The AMR Algorithm

The AMR algorithm is a general purpose mesh refinement scheme for producing very high resolution simulations of shock hydrodynamic phenomena. A full description of the algorithm is given by Quirk[15], and so here we shall merely attempt to impart its main features. Before proceeding it should be acknowledged that the foundations of the scheme lie with the work of Berger[1, 2].

The AMR algorithm employs a hierarchical system of grids. At the bottom of the hierarchy there is a coarse grid that delineates the computational domain. Additional tiers are added in order to refine this domain locally. The exact nature of the grid system may be visualized in the following manner. Imagine several sheets of squared graph paper with spacings $1, p, p.q, p.q.r, \ldots$ where p, q, r *etc.* are arbitrary integers, and suppose that these sheets are carefully stacked in ascending order of resolution such that the printed lines on successive sheets line up with one another. Now consider the following two rules for drawing rectangles on these sheets of paper. One, all lines must be drawn along the existing printed lines. Two, a rectangle drawn on one sheet of paper must be contained within one or more rectangles drawn on the sheet immediately below it. Note that this second rule does not apply to rectangles drawn

on the bottommost sheet. Given these two rules, an arbitrary computational grid would consist of all the squares contained by an arbitrary set of rectangles drawn on these stacked sheets. In practice the rectangular mesh patches may be distorted so as to form a body-fitted grid[4].

This hierarchical grid system provides a flexible means of discretizing a flow solution; each mesh cell contains a cell-centred projection of the flow solution. Additionally, the grid structure may be made to automatically adapt to an evolving flow. Thus it is possible to achieve the resolution associated with very fine meshes without incurring the expense of having to employ a fine mesh throughout the flow domain.

The AMR algorithm refines in time as well as space. More, but smaller time steps are taken on fine grids than on coarse grids. The hierarchical nature of the grid system allows the different sized time steps to be interleaved such that the simulation remains time accurate. Therefore, in contrast to other mesh refinement schemes, the presence of a few extremely fine mesh cells in one part of the flow domain does not have an adverse affect on the rate at which the rest of the flow solution may be advanced. This temporal refinement strategy should not be confused with the local time stepping strategy that is often used to accelerate convergence during steady-state calculations. Here we are concerned with computing time accurate solutions to unsteady flow problems.

An important feature of the AMR algorithm is that it places no special constraints on the basic numerical method used to integrate the discretized flow solution. For the algorithm contains machinery which allows each mesh patch to be integrated independently of every other mesh patch. So, in principle, any cell-centred solver developed for a single quasi-rectangular mesh could form the basis of the flow integration process. To date we have incorporated three different schemes to integrate the Euler and the Navier-Stokes equations, and one scheme for detonation flows. For the calculations presented in section 5, we used Toro's[16] hybridized Riemann solver to integrate the Euler equations. However, for the purposes of this paper the innermost workings of the integration process are unimportant.

4.1 Modifications

The following minor modifications had to be made to the AMR algorithm in order that it could take advantage of the cartesian boundary scheme. Firstly, every time the computational grid is adapted, so miscellaneous cell-type information must be

[4]Given that we now favour a cartesian boundary approach, this option is redundant.

gathered on those levels for which the grid has changed. Since a grid level merely consists of one or more rectangular mesh patches, the procedure which co-ordinates the gathering process is quite straightforward. The method described in section 3 is simply applied to each mesh patch in turn. However, it is worth noting one small detail that greatly improves the efficiency with which the cell-type information may be gathered, especially when several hundred patches need to be processed. The four control points used to define a cubic-Bézier curve form a convex hull, that is, the Bézier curve lies wholly within the quadrilateral formed from joining up the control points. Therefore, when processing a specific mesh patch, there is no need to trace a Bézier curve if its convex hull fails to overlap the mesh, for no intersection points will be found. Consequently, for the sake of a simple check just prior to tracing a curve, much wasted effort can be avoided.

Given the cell-type information, the procedure used to integrate a single mesh patch is but a slightly modified version of that used in the original body-fitted code. Note that the integration process may be split up by function into three parts: first, a set of unit area, interface fluxes is computed using some favoured numerical scheme; second, a series of surface flux integrals are computed so as to produce a set of residuals; third, the flow solution is updated by adding the residuals to the current solution. Dealing with the first part, the procedure used to compute a unit area, interface flux now monitors the cell types on either side of the interface. One of three situations is possible: both cells are solid, in which case no flux exists, so no processing is required; just one of the cells is solid, so a fictitious state is produced via a local reflection before proceeding on with the original routine; the combination of cell types is such to allow the straightforward use of the original routine. As was the case with the body-fitted code, the procedure for calculating a flux is applied to each interface of the mesh in turn. Following this however, it is now necessary to compute an auxiliary set of fluxes. Running down the list of cut cells contained by the specific mesh in hand, unit area fluxes are computed for those interfaces which form part of a solid surface. Again, so as to be able to use the standard flux formulation, a fictitious state is first produced via a local reflection before using the original flux routine.

Following the calculation of both sets of interface flux, it is a simple matter to perform a surface flux integral for every non-solid cell contained by the mesh. The procedure for an uncut cell is no different to that used by the body-fitted code except for the streamlining that is possible now that cells are square rather than arbitrary quadrilaterals. While the procedure to deal with cut cells is also straightforward,

it is somewhat tedious to code, for each cell-type needs to be handled separately. Note that the two intersection points which were used to determine the type for a cell also fix the respective areas to be used for the different fluxes in the summation of the surface flux integral. For the body-fitted code, the residuals arising from the flux summations could be used directly to update the flow solution, but for the cartesian boundary code they must be post-processed so as to ensure the stability of the integration process. Running through the lists of combination cells, the residuals for the cells in any one list are replaced by their volume weighted average. Again, it is worth emphasizing that this step should not be construed as an averaging procedure, as was described in section 2, it is just a convenient way in which to compute the residual for some odd-shaped cell. After this post-processing, the residuals are used in the usual manner to update the flow solution.

Finally, it should be noted that the comparative lack of detail given in this section merely reflects the mundaneness of the modifications.

5 Numerical Results

To demonstrate the effectiveness of our scheme we now present results for three test problems. Each calculation employed a three level adaptive grid. A coarse grid of 120 by 80 cells was used to delineate the computational domain, and a further two grid levels were used to resolve flow details. The spatial refinement factor between each grid level was 4, thus the resolution of the adaptive grid was nominally equivalent to that of a uniform mesh of some 1920 by 1280 cells. The computations were performed on a Silicon Graphics workstation.

Our first set of results come from the simulation of a planar shock wave reflecting from a circular cylinder. The shock Mach number, M_s, is equal to 2.81, and the gas is assumed to be ideal with a ratio of specific heats, γ, equal to 1.4. This problem has been used by many researchers to validate their shock capturing codes, for the computational results at one particular time instant may be readily compared against the Schlieren photograph presented by Bryson and Gross[4]. Figure 7 shows the density contours for our simulation at this time instant. Note we have taken advantage of the fact that the flow field is symmetric and have computed the flow about just one half of the cylinder. The similarity between these results and the Schlieren photograph is quite striking. All the salient features of the flow field are well resolved. The resolution of the contact discontinuity, vortex and vortex stem are particularly impressive. Indeed, most published results for this test problem simply

fail to show these features. Lastly, we note that the present results are comparable, if not superior, to those obtained using the original body-fitted version of our AMR code[15].

For the second test problem we chose to simulate the reflection of a planar shock wave over a double wedge, since this poses a sterner test than the ubiquitous single wedge calculation. A comprehensive numerical study of the different types of basic flow pattern that can evolve for this type of problem has been presented by Itoh *et al*[12]. Here we reproduce one of their examples, namely the case for which M_s is 2.16 and the wedge angles are 20^o and 55^o. Figure 8 shows density contours for our simulation at a stage in the evolution of the flow corresponding to that of the holographic interferogram shown on page 1164 of [12]. Unlike Itoh's simulation which is woefully under resolved, our results faithfully reproduce the salient features of the interferogram. However there is one anomaly. One of the contact discontinuities within the simulation exhibits a fully blown Kelvin-Helmholtz instability, the corresponding feature within the interferogram merely exhibits the early stages of such an instability. There are two plausible explanations for this discrepancy. First, the simulation did not include viscous effects and so would not be expected to accurately reproduce the growth of a Kelvin-Helmholtz instability. Second, the mechanism by which contact discontinuities are steepened was overly compressive. In other words, too much anti-diffusion was added and this accelerated the growth of the instability. Either way the cartesian boundary procedure is blameless and so we feel justified in ignoring this anomaly here.

Finally, we present results for a somewhat frivolous test problem. Figure 9 shows two snapshots from the interaction of a planar shock wave with the letters AMR. Although frivolous, this test amply demonstrates that the scheme can indeed cope with arbitrarily complex, two-dimensional shapes. And so it debunks the widely held view that problems which involve awkward geometries are the exclusive preserve of unstructured grid schemes. At this point it is worth emphasizing that the same code was used for all three test problems presented in this section. It simply ran with different sets of input data, each of which merely prescribed the outline of the relevant solid surfaces and the strength of the incident shock wave.

6 Conclusions

A simple method has been developed whereby solid wall boundary conditions may be imposed on a cartesian mesh. This method has been combined with an adaptive mesh refinement scheme to give a powerful algorithm for simulating shock hydrodynamic flows around arbitrarily complex, two-dimensional bodies. Results are presented which clearly demonstrate that the new algorithm can match, not only the accuracy of results produced using body-fitted grids, but also the geometric flexibility exhibited by unstructured grid schemes. As such, the algorithm constitutes a competitive alternative to existing methods for simulating flows which involve awkward geometries.

The performance of our algorithm results more from good management than from sophisticated numerics. Indeed, our scheme could be viewed as being humdrum, for no one component is extraordinary. However, this state of affairs is merely a reflection of Occam's razor; *entities are not to be multiplied beyond necessity.*

As it stands, the algorithm forms a rounded piece of work. However, it could be usefully extended in at least one direction. Namely, it should be possible to extend the scheme so as to allow bodies to move relative to the mesh. This would open up many new applications, such as modelling sabot and store release phenomena.

Finally, the resolution of the simulations presented in this paper is sufficient to highlight the limitations of using the Euler equations to model shock hydrodynamic flows. In particular, the modelling of contact discontinuities calls for the use of the Navier-Stokes equations. Whilst the inclusion of viscous terms into the framework of our cartesian boundary scheme is straightforward, given the lack of any notion of a preferred direction, such a scheme would be limited to low Reynolds number flows. Further work would be required to extend our methodology to high Reynolds numbers.

Acknowledgements

I would like to thank my former supervisor, Prof. P.L.Roe, and my colleagues: Dr. E.F.Toro, Mr. J.Pike and Dr. J.A.Edwards, for the discussions which laid the foundations of this work.

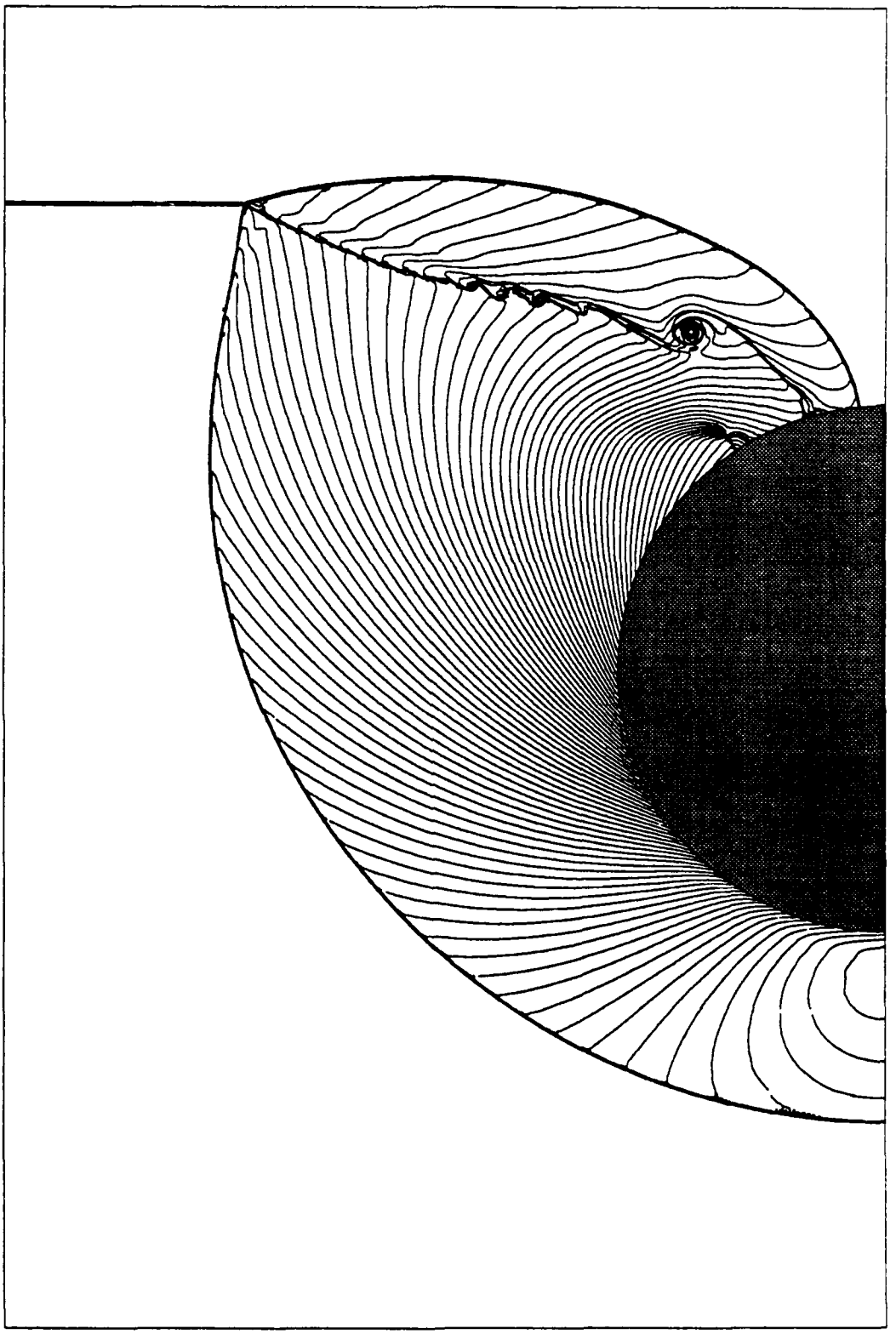


Figure 7: Density contours, and corresponding computational grid, for one snapshot from the interaction of a planar shock wave with a circular cylinder; cf. Schlieren photograph presented by Bryson and Gross[4].

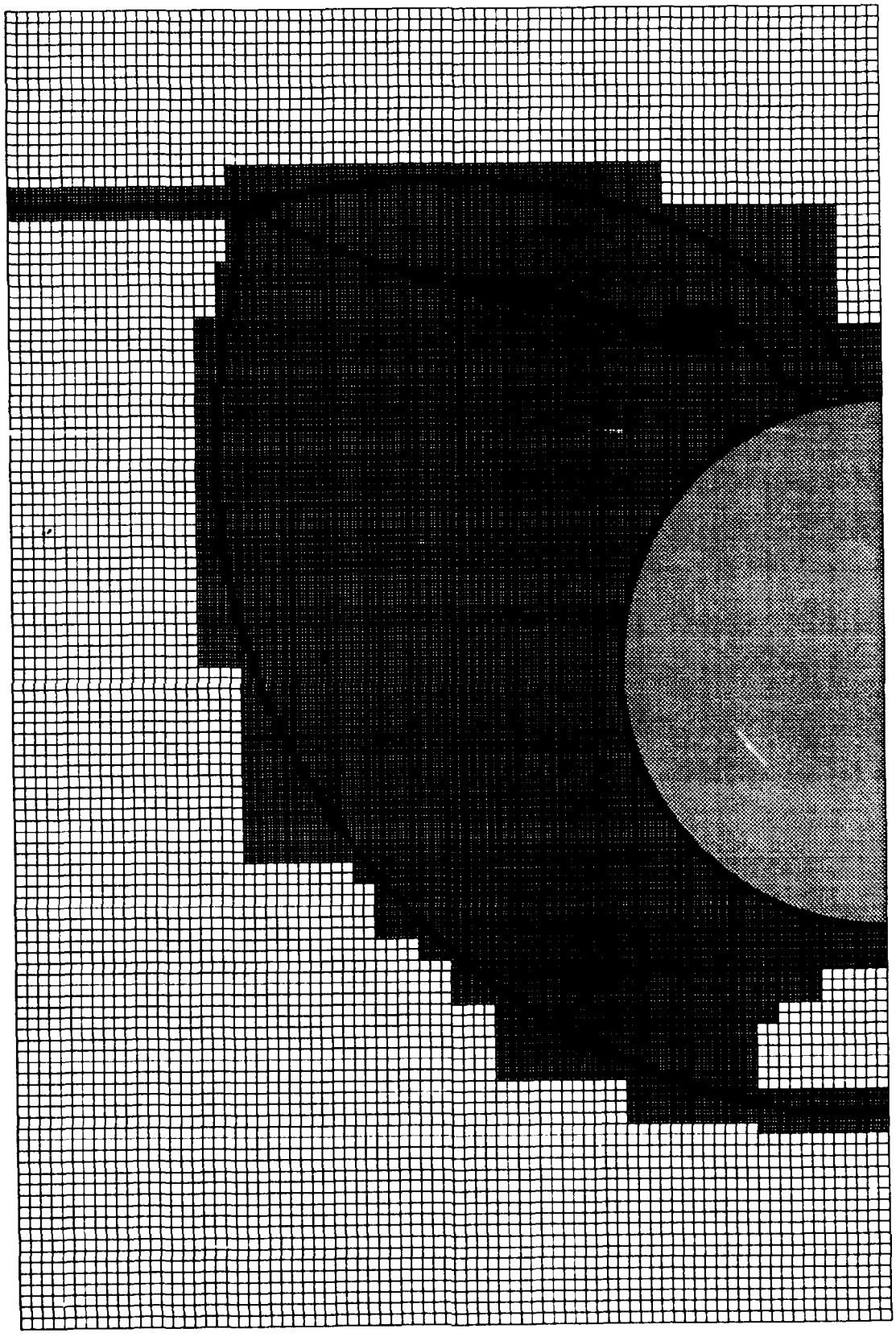


Figure 7: For caption, see facing page.

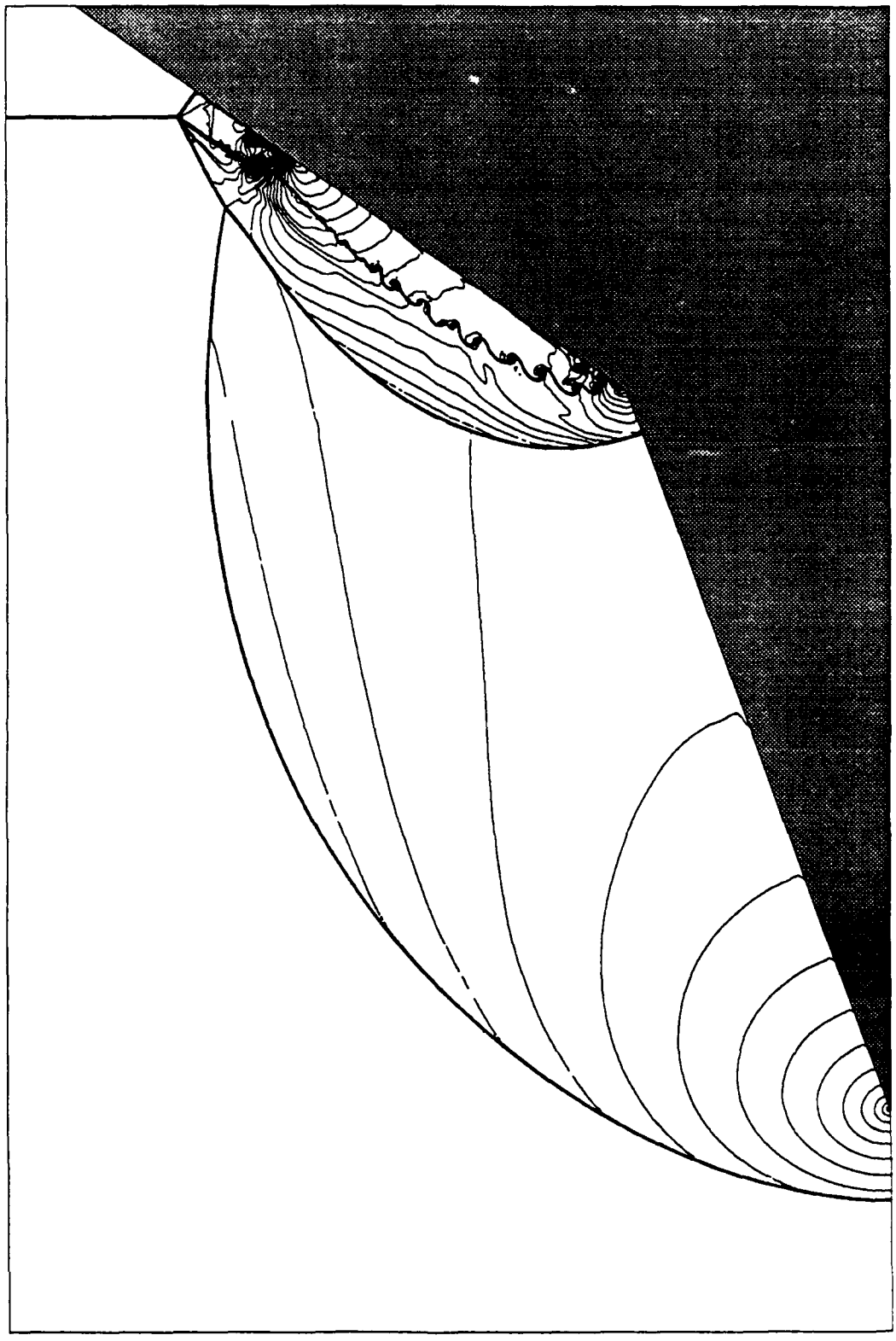


Figure 8: Density contours, and corresponding computational grid, for one snapshot from the interaction of a planar shock wave with a double wedge; cf. holographic interferogram presented by Itoh *et al*[12].

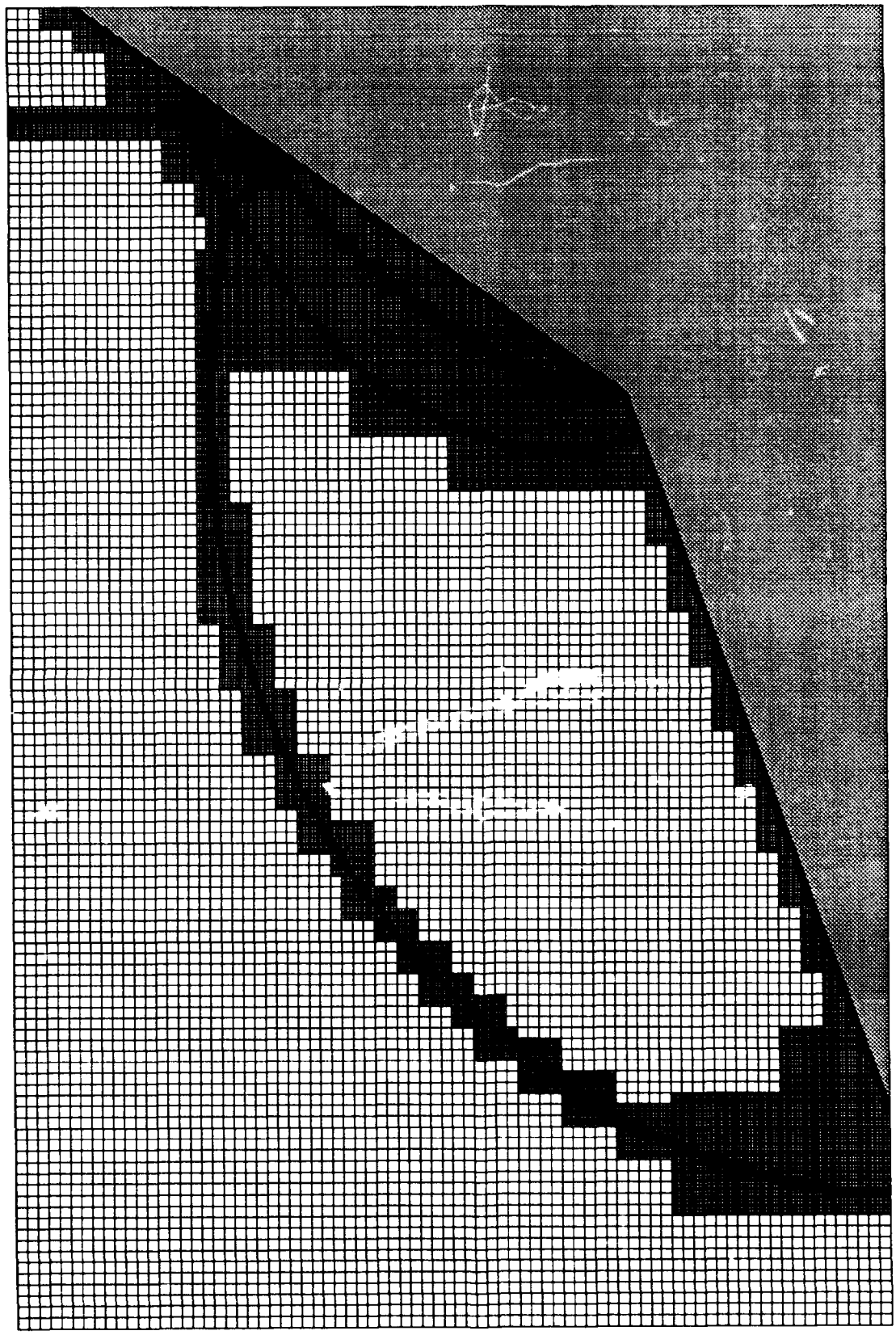


Figure 8: For caption, see facing page.

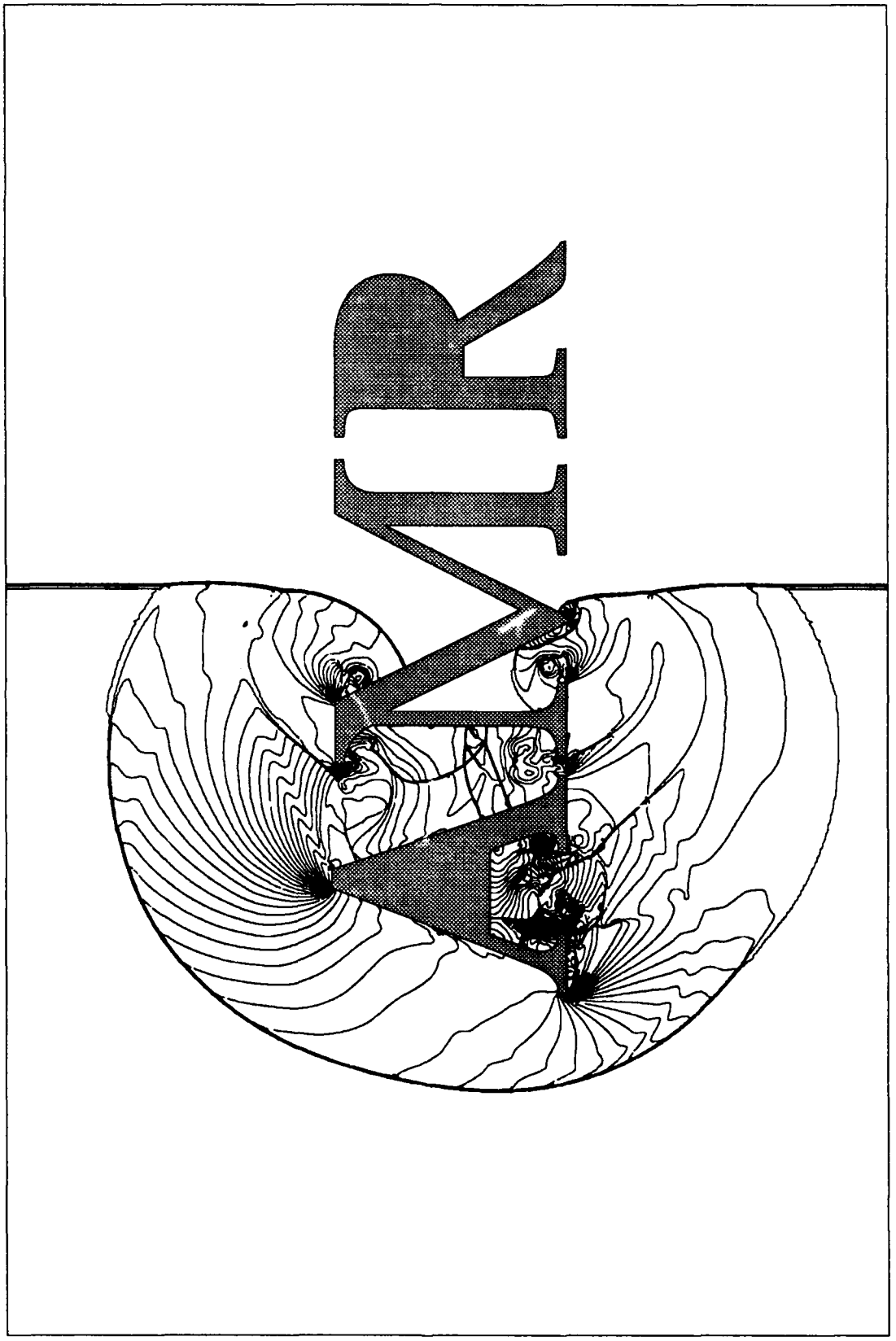


Figure 9: Two snapshots from the interaction of a planar shock wave with the letters AMR. Both plots show density contours.

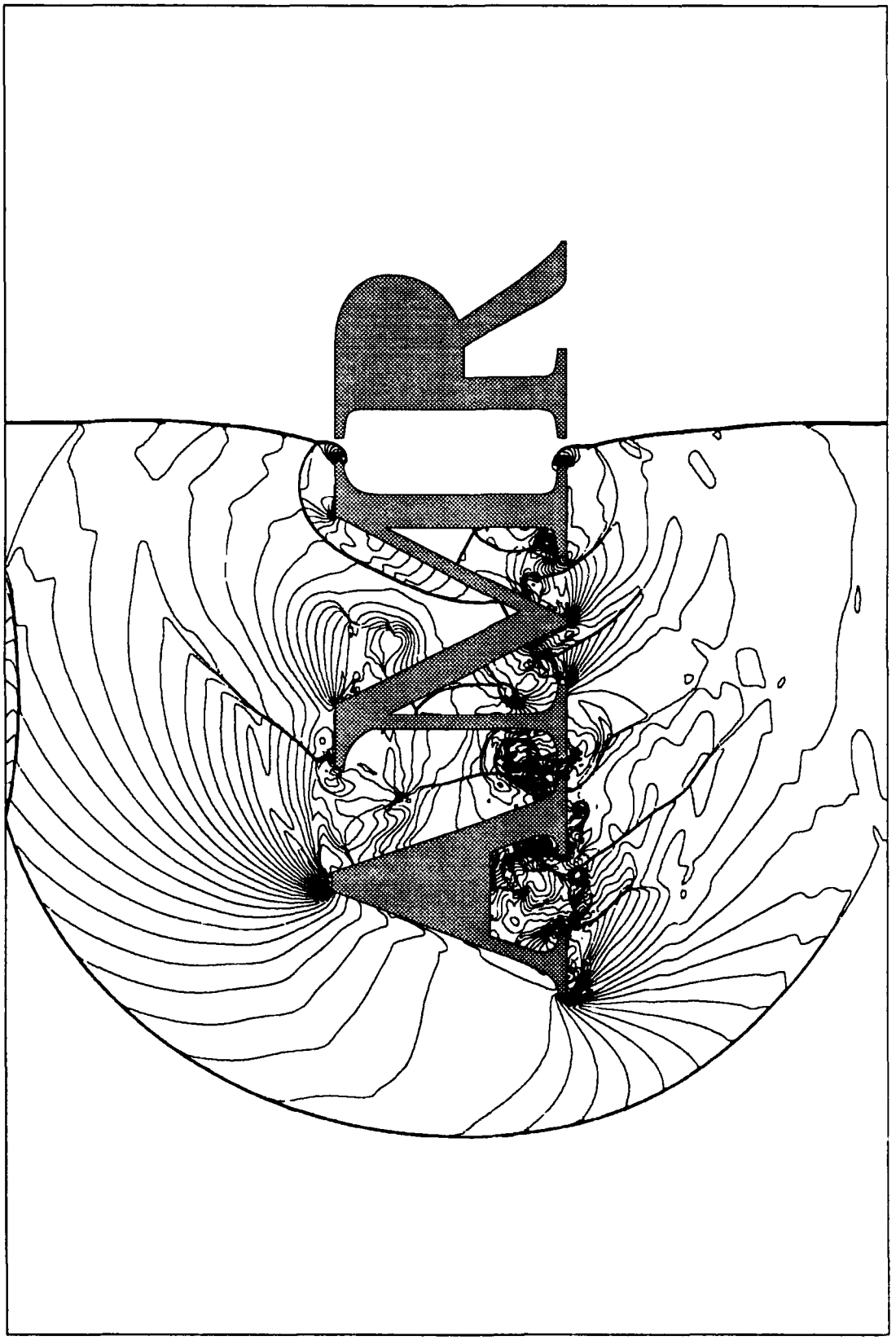


Figure 9: For caption, see facing page.

References

[1] M.J.BERGER, *Adaptive mesh refinement for hyperbolic partial differential equations.* Ph.D. thesis, Computer Science Dept., Stanford University (1982).

[2] M.J.BERGER AND P.COLLELA, *Local adaptive mesh refinement for shock hydrodynamics.* J. Comput. Phys., **82**(1989), pp. 67–84.

[3] M.J.BERGER AND R.J.LE VEQUE, *An adaptive cartesian mesh algorithm for the Euler equations in arbitrary geometries.* AIAA Paper 89-1930-CP (1989).

[4] A.E.BRYSON AND R.W.F.GROSS, *Diffraction of strong shocks, by cones, cylinders, and spheres.* J. Fluid Mech., Vol. **10**(1961), pp. 1–16.

[5] 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.

[6] D.DE ZEEUW AND K.G.POWELL, *An adaptively refined cartesian mesh solver for the Euler equations.* AIAA Paper 91-1542 (1991).

[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] J.D.FOLEY, A.VAN DAM, S.K.FEINER AND J.F.HUGHES, *Computer graphics, principles and practice.* 2nd edn. Addison Wesley (1990).

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

[10] *Graphics Gems.* Edited by A.S.Glassner, Academic Press (1990).

[11] G.D.VAN ALBADA, B.VAN LEER AND W.W.ROBERTS, *A comparative study of computational methods in cosmic gas dynamics.* Vol. **108**(1982), No. 1, pp. 76–84.

[12] K.ITOH, K.TAKAYAMA AND G.BEN-DOR, *Numerical simulation of the reflection of a planar shock wave over a double wedge.* Int. J. for Numer. Meth. in Fluids, Vol. **13**(1991), pp 1153–1170.

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

[14] 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).

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

[16] E.F.TORO, *A linearised Riemann solver for the time-dependent Euler equations of gas dynamics.* Proc. Roy. Soc. London, A **434**(1991), pp. 683–693.

[17] B.VAN LEER, *Towards the ultimate conservative difference scheme,* II. *Monotonicity and conservation combined in a second-order scheme.* J. Comput. Phys., **14**(1974), pp. 361–376.

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

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 1992	Contractor Report

4. TITLE AND SUBTITLE
AN ALTERNATIVE TO UNSTRUCTURED GRIDS FOR COMPUTING GAS DYNAMIC FLOWS AROUND ARBITRARILY COMPLEX TWO-DIMENSIONAL BODIES

5. FUNDING NUMBERS
C NAS1-18605
WU 505-90-52-01

6. AUTHOR(S)
James J. Quirk

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Institute for Computer Applications in Science and Engineering
Mail Stop 132C, NASA Langley Research Center
Hampton, VA 23665-5225

8. PERFORMING ORGANIZATION REPORT NUMBER
ICASE Report No. 92-7

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23665-5225

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
NASA CR-189612
ICASE Report No. 92-7

11. SUPPLEMENTARY NOTES
Langley Technical Monitor: Michael F. Card
Final Report
Submitted to Computers & Fluids

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Unclassified - Unlimited

Subject Category 02,64

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Within the shock-capturing community, the need to simulate flows around geometrically complex bodies has resulted in an inexorable shift away from schemes which employ body-fitted grids to schemes which employ unstructured grids. Although unstructured grids are undeniably effective, in view of the increasing reliance placed on computational results, such a wholesale shift in mentality should give cause for concern. The concept of using several computer codes to cross check numerical results becomes ill-founded if all codes follow the same methodology. In this paper we describe an alternative approach for dealing with arbitrarily complex, two-dimensional geometries, the so-called cartesian boundary method.

Conceptually, the cartesian boundary method is quite simple. Solid bodies blank out areas of a background, cartesian mesh, and the resultant cut cells are singled out for special attention. However, there are several obstacles that must be overcome in order to achieve a practical scheme. We present a general strategy that overcomes these obstacles, together with some details of our successful conversion of an adaptive mesh algorithm from a body-fitted code to a cartesian boundary code.

14. SUBJECT TERMS	15. NUMBER OF PAGES
complex geometries; cartesian grids; mesh adaption, unsteady flows	31
	16. PRICE CODE A03

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		