INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids 2004; **46**:1099–1125
Published online 20 October 2004 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/fld.780

Two-dimensional anisotropic Cartesian mesh adaptation for the compressible Euler equations

W. A. Keats*,† and F.-S. Lien

Department of Mechanical Engineering, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada

SUMMARY

Simulating transient compressible flows involving shock waves presents challenges to the CFD practitioner in terms of the mesh quality required to resolve discontinuities and prevent smearing. This paper discusses a novel two-dimensional Cartesian anisotropic mesh adaptation technique implemented for transient compressible flow. This technique, originally developed for laminar incompressible flow, is efficient because it refines and coarsens cells using criteria that consider the solution in each of the cardinal directions separately. In this paper, the method will be applied to compressible flow. The procedure shows promise in its ability to deliver good quality solutions while achieving computational savings.

Transient shock wave diffraction over a backward step and shock reflection over a forward step are considered as test cases because they demonstrate that the quality of the solution can be maintained as the mesh is refined and coarsened in time. The data structure is explained in relation to the computational mesh, and the object-oriented design and implementation of the code is presented. Refinement and coarsening algorithms are outlined. Computational savings over uniform and isotropic mesh approaches are shown to be significant. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: anisotropic mesh; Euler equations; shock waves; Cartesian geometry; transient adaptation

1. INTRODUCTION

This paper focuses on the use of the anisotropic Cartesian mesh adaptation technique to improve the resolution of numerical flow simulations governed by the two-dimensional Euler equations. In particular, flows with strong transient shock waves are chosen to test the ability of the refinement algorithm to handle moving discontinuities.

*Correspondence to: W. Andrew Keats, Department of Mechanical Engineering, University of Waterloo, 200 University Avenue West, Waterloo, Ontario N2L 3G1, Canada.
†E-mail: wakeats@alumni.uwaterloo.ca

Received 2 April 2004
Revised 2 August 2004

1.1. Governing equations

The Euler equations provide a model for the inviscid compressible flow of a homogeneous fluid in subsonic and supersonic regimes [1]. They can be used to numerically predict the locations and shapes of shock waves that occur in regions of the flow where the effects of viscosity are negligible.

In two space dimensions, the Euler equations are

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial \mathbf{f}}{\partial x} + \frac{\partial \mathbf{g}}{\partial y} = 0 \tag{1}$$

where

$$\mathbf{u} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho e_T \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (\rho e_T + p)u \end{bmatrix}, \quad \mathbf{g} = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ (\rho e_T + p)v \end{bmatrix}$$

The variable $\mathbf{u}$ is the vector of conserved variables; mass, momentum and energy, all per unit volume. The pressure p is obtained using an equation of state for ideal gases:

$$p = (\gamma - 1)(\rho e_T - \tfrac{1}{2}\rho(u^2 + v^2)) \tag{2}$$

The variables $\mathbf{f}$ and $\mathbf{g}$ represent horizontal and vertical mass, momentum and energy fluxes, plus their respective pressure contributions.

1.2. Mesh adaptation

Qualitatively speaking, the governing equations are solved over a mesh consisting of a multitude of tiny adjacent finite volumes (otherwise known as *cells*). Each cell maintains corresponding $\{\rho, p, u, v\}$ values, usually located at the centroid of the cell, and the overall solution consists of the field of $\{\rho, p, u, v\}$ encompassed by the mesh. Thus, the resolution of the numerical solution is limited by the distance between adjacent finite volumes.

Mesh adaptation serves to selectively modify the mesh layout so that in regions of the flow where high resolution is required to discern flow features, more cells are added. Likewise, in smooth regions of the flow, cells are removed in order to increase the computational efficiency. The total number of cells in the computational mesh does not necessarily need to remain constant, as long as it does not grow beyond the memory capacity of the computer. Isotropic mesh adaptation refines areas of the mesh by subdividing cells into smaller cells of equal aspect ratio. Anisotropic mesh adaptation does not necessarily preserve aspect ratio while subdividing and joining cells.

1.3. Scope

Isotropic Cartesian mesh adaptation has been presented in References [2, 3] and applied to steady and transient compressible flow. However, anisotropic Cartesian mesh adaptation has only been used to solve transient incompressible [4] and steady compressible [5] flows.

All of the test cases considered in this paper are transient and involve strong shock waves. The refinement and coarsening algorithms presented in Reference [4] cannot be directly applied to such test cases and must be modified to meet the mesh smoothness requirements of compressible flow.

2. NUMERICAL METHOD

2.1. Finite volume discretization

In this paper we consider unstructured two-dimensional anisotropic Cartesian grid cells, such as the one shown in Figure 1. The conserved variables $\mathbf{u}$ are stored at the cell centres, and the face fluxes $\mathbf{f}$ and $\mathbf{g}$, stored at the faces, are indexed by cardinal direction $\{\mathrm{N}, \mathrm{S}, \mathrm{E}, \mathrm{W}\}$ and position (denoted by subscript $\{0, 1\}$).

For the general case of a Cartesian grid cell with a maximum of two faces in each of the cardinal directions, integration of the Euler equations over a control volume yields the discrete equation:

$$\mathbf{u}^{n+1} = \mathbf{u}^n - \Delta t \frac{1}{\Delta x \Delta y} \underbrace{\left[\sum_{i=0}^{1} (l_{\text{face}}\mathbf{f})_i^{\mathrm{E}} - \sum_{i=0}^{1} (l_{\text{face}}\mathbf{f})_i^{\mathrm{W}} + \sum_{i=0}^{1} (l_{\text{face}}\mathbf{g})_i^{\mathrm{N}} - \sum_{i=0}^{1} (l_{\text{face}}\mathbf{g})_i^{\mathrm{S}} \right]}_{[\cdot]=\mathscr{F}(\mathbf{u})} \tag{3}$$

where Δx and Δy are the width and height of the cell, respectively, and l_{face} refers to the length of the face in position i. In the more specialized case of a uniform, structured Cartesian grid, Equation (3) reduces to

$$\mathbf{u}^{n+1} = \mathbf{u}^n - \frac{\Delta t}{\Delta x}(\mathbf{f}_0^{\mathrm{E}} - \mathbf{f}_0^{\mathrm{W}}) - \frac{\Delta t}{\Delta y}(\mathbf{g}_0^{\mathrm{N}} - \mathbf{g}_0^{\mathrm{S}}) \tag{4}$$

Equation (4) is first-order accurate in time. Higher-order extension is discussed in Section 2.4.3.

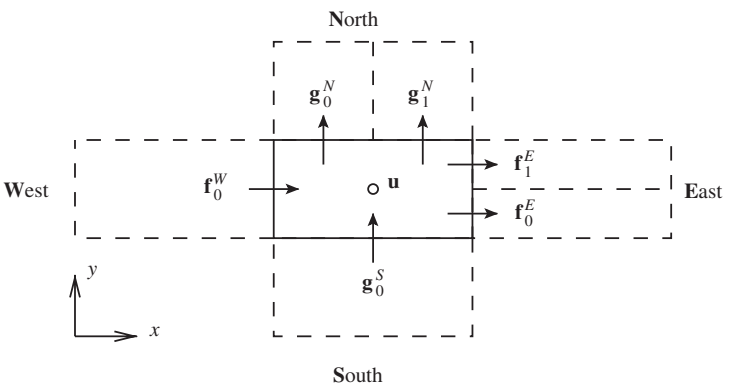

Figure 1. A sample Cartesian grid cell with multiple faces and neighbours.

2.2. Boundary conditions

2.2.1. Walls. Any cell face may be specified as a *wall* boundary condition, in which case the flux through the face is modified by setting the normal component of the velocity to zero. For horizontal wall-faces, $v = 0$; for vertical walls, $u = 0$. This leaves only a pressure term in the momentum flux; for simplicity, this term is set equal to the cell-centred pressure, so that:

$$p(x_{i+1/2}, y, t) \approx p(x_i, y, t^n) + O(\Delta x) + O(t - t^n) \tag{5}$$

is a constant extrapolation of pressure that is first-order accurate in space [6]. Linear and higher-order extrapolation may be used instead.

2.2.2. Inflow and outflow. All inflow and outflow boundary conditions used in the test cases described in this paper are supersonic. At inlets, all primitive flow variables are specified and interface fluxes calculated accordingly. At outlets, no variables require specification; fluxes are calculated using the cell-centred conserved variables **u** from the previous time step.

2.3. First-order numerical scheme

The AUSM$^+$ scheme of Liou [7] was chosen as the base numerical scheme for the present work. It is an extension of the original advection upstream splitting method (AUSM) scheme of Liou and Steffen [8] in which the flux vector, divided into convective and pressure components, is split based on Mach number. The division occurs in the momentum component of the flux, but not in the energy component.

This scheme was used throughout the research for the following reasons:

1. It is a more recent flux vector splitting that overcomes the disadvantages of earlier alternatives, yet it has already been proved reliable for a range of different flow regimes (see, for example, Reference [9]), and is simple to implement in code.
2. The order of accuracy of the scheme was found by Ripley [10] to be 1.81 on a uniform Cartesian mesh.

2.4. Higher-order extension

Spatial and temporal extension of the numerical method to second- and higher-order accuracy provides a way of superlinearly improving the accuracy of the solution as cell sizes and time steps are reduced.

In all of the numerical methods discussed previously, cell data are assumed piecewise-constant; the flux at the face is generated by a cell-centred conserved variable **u**. This lack of variation in **u** introduces dissipative errors into the solution (see Reference [11] for a discussion of dissipative and dispersive errors) which result in the unwanted smearing of flow features such as shock and contact discontinuities.

2.4.1. Spatial extension. In order for a finite-volume numerical method to achieve second-order accuracy, gradient information must be used in the calculation of the fluxes at cell interfaces. In van Leer's monotone upstream-centred scheme for conservation laws (MUSCL)

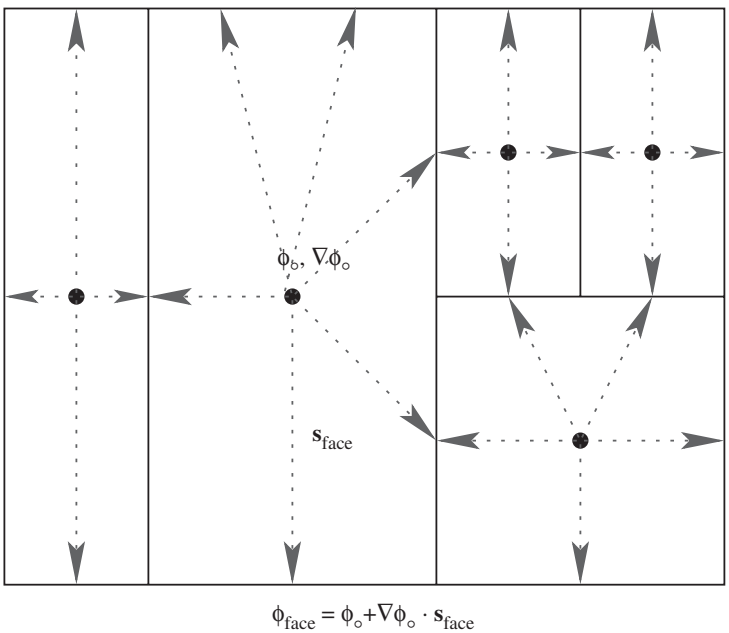

Figure 2. Cell-centred ϕ are extrapolated to cell faces.

approach, piecewise-constant cell-centred data is modified by higher-order components of the Taylor series approximation to **u** at the cell faces [12]. Specifically, for a second-order accurate solution, the cell-centred **u** must be extrapolated to the cell faces using $\nabla\mathbf{u}$ in order to calculate $\mathbf{f}^{\pm}$ and $\mathbf{g}^{\pm}$. This extension to the numerical method is vital when using irregular meshes, since first-order finite differences lose accuracy when grid spacing is non-uniform.

Figure 2 demonstrates a typical part of an anisotropic Cartesian mesh, in which cell-centred ϕ values (individual components of **u**) are linearly extrapolated to face centres via the gradient, $\nabla\phi$.

2.4.2. Gradient limiters. First-order numerical schemes are highly dissipative; second and higher order are much less so. Gradients calculated based on information across flow discontinuities, such as shock and contact waves, are invalid, and may lead to the catastrophic destabilization of the solution. For example, fluxes calculated based on unrealistic gradient information may yield negative pressures and densities at the next time step.

In order to overcome these problems while maintaining second-order accuracy throughout the majority of the domain, the solution gradients must be modified in the presence of discontinuities and other sharp flow features. In the present code, gradients are modified using either the *minmod* limiter or the *superbee* limiter [12].

The use of limiters to smooth gradient information does not guarantee the stability of the solution in two or more dimensions. However, it provides great improvement over unlimited solutions, most of which do not converge (they result in negative pressure and density) at high Mach numbers.

2.4.3. Temporal extension. In this work, a formally second-order accurate three-stage Runge–Kutta time-stepping scheme was adopted for the following reasons:

1. The first-order time-stepping procedure of Equation (4) is highly sensitive to CFL number when combined with the AUSM$^+$ scheme. Using the three-stage Runge–Kutta technique allows improved stability at CFL numbers approaching unity.
2. For high Mach numbered test cases (Mach 2 and above), additional dissipation is needed to maintain positivity of density and pressure. By using a combination of first- and second-order spatial accuracy through different time steps, the solution can approach second-order accuracy in space while remaining stable.

This technique, outlined in Reference [11], is reproduced below:

$$\begin{aligned}
\mathbf{u}^{(*)} &= \mathbf{u}^n - \Delta t\, \mathscr{F}(\mathbf{u}^n) \\
\mathbf{u}^{(**)} &= \frac{1}{2}[\mathbf{u}^n + \mathbf{u}^{(*)}] - \frac{\Delta t}{2}\mathscr{F}(\mathbf{u}^{(*)}) \\
\mathbf{u}^{n+1} &= \frac{1}{2}[\mathbf{u}^n + \mathbf{u}^{(**)}] - \frac{\Delta t}{2}\mathscr{F}(\mathbf{u}^{(**)})
\end{aligned} \tag{6}$$

where $\mathscr{F}(\mathbf{u})$ is the summation of the fluxes through all cell faces, as in Equation (3).

3. CARTESIAN ANISOTROPIC ADAPTATION TECHNIQUE

The adaptation method discussed here is based on a method that was originally developed by Ham *et al.* [4]. It is an unstructured face-based method that can transiently refine and coarsen grid cells. Ham *et al.*, applied the method to two- and three-dimensional incompressible flow at Reynolds numbers below 150. The main advantage of using anisotropic grid refinement over isotropic refinement is the fact that memory and computational savings can be realized when the flow gradients are anisotropic in the x- and/or y-directions. Even if anisotropy is present away from these cardinal directions, the method still yields computational savings over an isotropic technique.

The adaptation method and criteria described in this section are independent of the choice of numerical method, provided that the method has a compact stencil and can be written in conservation form, or in other words, as the updating of cell-centred **u** based on interface **f** and **g**. The data structure and algorithm are similar to those presented in Reference [4], but several key differences allow the method to be applied to transient compressible flow. Specifically, Algorithms 3.1–3.4 place additional limits on the refinement levels of neighbouring cells in order to keep the mesh relatively smooth. Also, coarsening and refinement are performed less aggressively than in Reference [4] in order to minimize the effect of errors introduced in the extrapolation of cell-centred data to new cell centres.

3.1. Data structure

3.1.1. Object and class definitions. The present code is object-oriented, with individual classes defined for computational cells and interfaces, as well as their encompassing mesh, as shown in Figure 3.

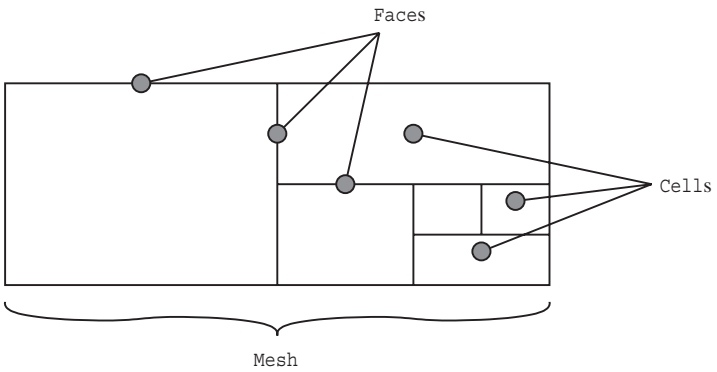

Figure 3. Classes of objects found in the computational mesh.

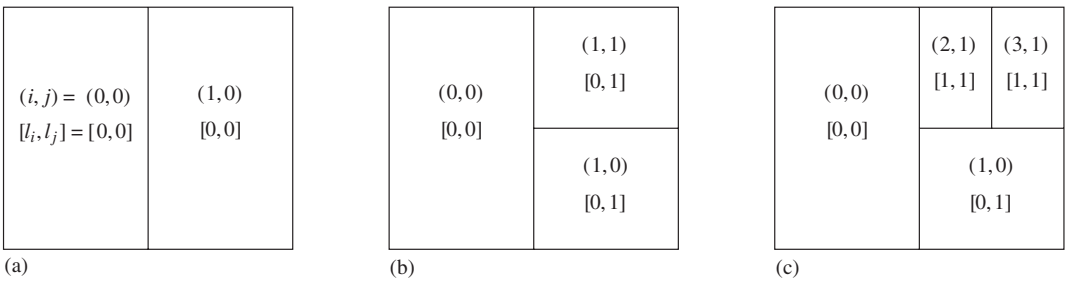

Figure 4. Cell indexing system used in the present code: (a) initial mesh (2 cells); (b) after a y-refinement; and (c) after a x-refinement.

In Reference [4], Ham *et al.*, use an (i, j) indexing scheme to keep track of cell (x, y) locations. The present code maintains (x, y) locations explicitly, but also retains the (i, j) indexing scheme in order to simplify the mesh adaptation process.

Figure 4 illustrates the nature of cell indexing. The (i, j) indices correspond to the indices that would be needed if the mesh were structured, with each cell at the same refinement level. The $[l_i, l_j]$ indices denote the refinement level of the cell; the computational mesh is initially constructed from cells whose refinement levels are zero. It is important to note that whereas the original indexing system of Reference [4] was based around a single origin cell of indices $(0, 0)$ and refinement level $[0, 0]$, the system has been modified to permit multiple initial cells (which aids in the formation of backward and forward steps) with refinement levels of $[0, 0]$, but non-zero indices.

The cell coarsening algorithm uses the (i, j) indices to determine suitable partner cells; in doing so, it helps to maintain the smoothness of the mesh by avoiding 'brick wall' formations, illustrated in Figure 5. Such irregular meshes were found to cause serious degradation in the solution quality.

In terms of variable storage, `Cell` and `Face` objects are required to keep track of their respective conserved variables and fluxes, as well as standard geometric parameters required

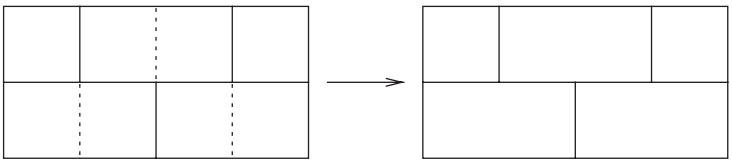

Figure 5. The (i, j) indexing system allows the algorithm to avoid such irregular coarsening.

Table I. Class definitions.

`Cell`	`Face`	`Mesh`
$\mathbf{u}$	$\mathbf{f}$	$\mathrm{CFL}_{\max}$
$\nabla\{\mathbf{u}\}_k$	(x, y)	$[l_i, l_j]_{\max}$
(x, y)	l	$[l_i.l_j]_{\min}$
$(\Delta x, \Delta y)$	bcFlags[]	dimension[]
(i, j)		
$[l_i, l_j]$		

Table II. Variable definitions.

Variable name	Definition
$\mathbf{u}$	Cell-centred conserved variable
$\nabla\{\mathbf{u}\}_k$	Gradient of kth $\mathbf{u}$ component
(x, y)	Cell/face-centred coordinates
$(\Delta x, \Delta y)$	Cell size
(i, j)	Cell index
$[l_i, l_j]$	Cell refinement level
$\mathbf{f}$	Face-centred flux
l	Face length
bcFlags[]	Boundary condition flags
$\mathrm{CFL}_{\max}$	Maximum CFL number within the mesh
$[l_i, l_j]_{\max}$	Maximum allowable cell refinement levels
$[l_i, l_j]_{\min}$	Minimum allowable cell refinement levels
dimension[]	Mesh layout parameters

in an unstructured, adaptive mesh. Tables I and II summarize the minimum necessary storage requirements for each class.

Flags must also be stored for the refinement and coarsening procedures; however, they can be implemented in many different ways and are thus left to the developer.

3.1.2. Object model. The relationships between `Cell`, `Face` and `Mesh` objects are shown in the unified modelling language (UML) aggregation diagram of Figure 6 (see, for example, Reference [13] for an introduction to UML and examples of aggregation diagrams). As described by Figure 3, the `Mesh` is composed of linked lists of individual `Cell` and `Face` objects. Linked lists are the most efficient data structures to use when objects must be dynamically added to and deleted from the list.

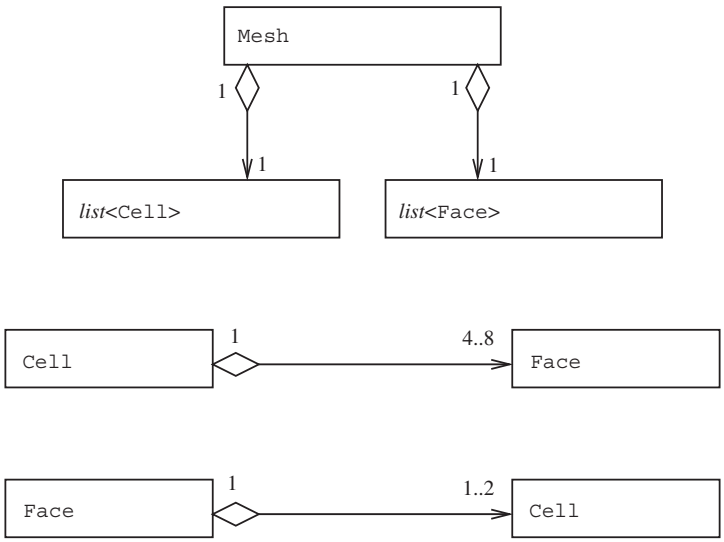

Figure 6. Class aggregation diagram.

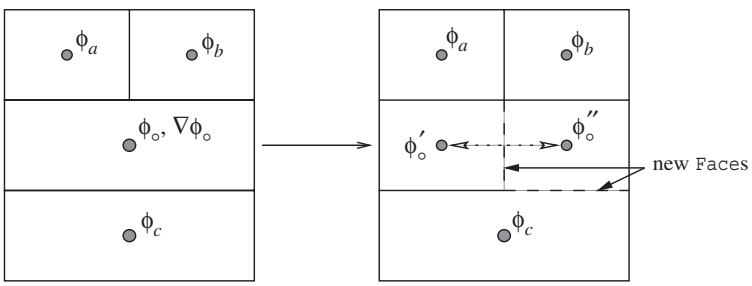

Figure 7. Cell refinement procedure.

3.1.3. Required object facilities. Without discussing fine details of the code, the classes must implement the functionality required by the refinement and coarsening processes:

Cell refinement: This process is outlined in Figure 7, and requires the following to be performed:

1. Create a new `Cell` and extrapolate $\phi_\circ$ to $\phi'_\circ$ and $\phi''_\circ$ using $\nabla\phi_\circ$.
2. Modify (x, y), $(\Delta x, \Delta y)$, (i, j), and $[l_i, l_j]$ of the refined `Cells` appropriately.
3. Create a new `Face` between the split `Cells`, and remap neighbouring `Cells`' faces appropriately. For example, in Figure 7, the `Cell` containing ϕ_c will need another `Face` to be added to its collection of North `Faces`.
4. Modify (x, y), l and bcFlags[] of any `Faces` that were shortened; supply appropriate (x, y), l and bcFlags[] to any newly created `Faces`.

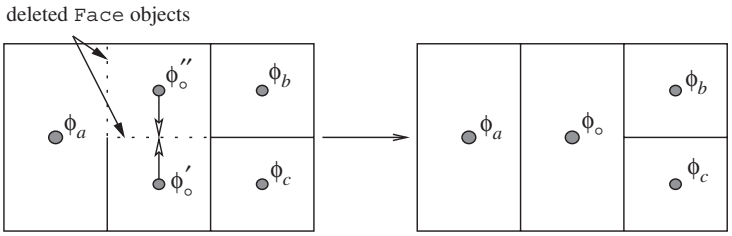

Figure 8. Cell coarsening procedure.

Cell coarsening: This process is outlined in Figure 8, and requires the algorithm to:

1. Choose two `Cells` of the same dimensions and appropriate indices (i, j); remove the upper (right) `Cell` and extend the lower (left) `Cell`.
2. Average $\phi'_\circ$ and $\phi''_\circ$ to obtain $\phi_\circ$.
3. Delete the upper `Cell` and intermediate `Face` from the `Mesh`'s list of `Cells` and `Faces`.
4. Modify (x, y), $(\Delta x, \Delta y)$, (i, j), and $[l_i, l_j]$ of the coarsened `Cell` appropriately.
5. Modify (x, y) and l of any `Faces` that must be lengthened; delete any `Faces` that were made redundant through the process of coarsening. For example, in Figure 8, the `Faces` beneath and to the left of $\phi''_\circ$ must be deleted.

3.2. *Refinement and coarsening algorithms*

Algorithms 3.1–3.4 reproduce in pseudo-code the x-refinement and x-coarsening algorithms as they are implemented in the present code. The y-refinement and y-coarsening algorithms are defined similarly. These algorithms differ from those of Ham *et al.* [4] in that additional restrictions are placed on the refinement levels of adjacent cells, as well as the fact that refinement proceeds from the largest to the smallest cells. It is important to note that the algorithms presented here do not explicitly account for refinement and coarsening next to boundary faces. Boundary faces cannot necessarily be ignored during the implementation of algorithms 3.1–3.4, since depending on the language used, checking the status of a cell on the other side of a boundary face may be an illegal operation. There are many possible ways of implementing boundary checks, so any necessary additions to the algorithms are best left to the programmer.

Algorithm notation: According to the `Face` and `Cell` class aggregation diagram of Figure 6, objects of one type carry specific references to neighbouring objects of another type.[‡] In the algorithms that follow, this relationship is described using the right arrow, '$\rightarrow$'. For example,

$$\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow \text{flaggedRefineX}$$

refers to `Face f`'s left-hand `Cell c`'s flag 'flaggedRefineX'.

[‡]e.g. in C++ a `Face` references its neighbouring `Cells` through pointer variables.

An arrow followed by an expression in braces means that the expression acts on the variable belonging to the object. For example,

$$\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow \{l_i = l_i - 1\}$$

means that `Face f`'s left-hand `Cell c`'s variable l_i must be decremented.

Loops and if-statements are terminated when the indent level of the pseudo-code block returns to the same level as the originating expression: 'end for' and 'end if' are implied. Comments are provided using slanted text enclosed by parentheses.[§]

Algorithm 3.1 Flag and refine `Cells` in the x-direction

Require: all refinement/coarsening flags set to *false*

for all `Cells c` **do**
  {*clear refinement & coarsening flags*}
  $\texttt{c} \rightarrow \text{flaggedRefineX} = \textit{false}$
  $\texttt{c} \rightarrow \text{flaggedCoarsenX} = \textit{false}$
  Calculate least-squares $\texttt{c} \rightarrow \nabla\phi$
  Calculate $\texttt{c} \rightarrow \Delta x_{\text{target}}$
for all `Faces f` **do**
  Calculate interface gradients
  if $\left[\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow \Delta x_{\text{target}} \leqslant \Delta x_L\right]$ OR $\left[\texttt{f} \xrightarrow{R} \texttt{c} \rightarrow \Delta x_{\text{target}} \leqslant \Delta x_R\right]$ **then**
    $\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow \text{flaggedRefineX} = \textit{true}$
    $\texttt{f} \xrightarrow{R} \texttt{c} \rightarrow \text{flaggedRefineX} = \textit{true}$
for all `Cells c` **do**
  Apply limiter to $\texttt{c} \rightarrow \texttt{Faces} \rightarrow \nabla\phi$ to generate new $\texttt{c} \rightarrow \nabla\phi$
{*go through `Cells` and refine level by level*}
for $i = l_{i,\min}$ to $l_{i,\max} - 1$ **do**
  for all `Cells c` **do**
    if $[\texttt{c} \rightarrow \text{flaggedRefineX} \equiv \textit{true}]$ **AND** $[\texttt{c} \rightarrow l_i \equiv i]$ **then**
      call refineX(c) {*see algorithm 3.2*}

3.3. Application of the algorithm to compressible flow simulations

The refinement and coarsening algorithms are not exactly the same as described in Reference [4]; the present code does not perform more than one iteration of cell coarsening for each refinement level. It was found that additional coarsening introduces numerical instability in areas of high gradient, while the memory savings are insignificant. Rather than calling the coarsen routine for each cell until all cells are at the largest size permitted by the mesh refinement criteria, the code makes $(l_{i,\max} - l_{i,\min})$ iterations through the code, coarsening the largest cells at first, and proceeding to the finest. Coarsening the largest cells first ensures that for a sweep of coarsening, any pair of cells are only joined once, which helps to maintain the solution quality.

[§]{*For example, this is a comment.*}

Algorithm 3.2 *boolean* refineX(`Cell c`)

```
1: for all neighbouring Cells c_nb ∈ Faces {N,S} do
2:    {refine Cells that are preventing c's refinement}
3:    if [c_nb → l_i ≡ l_i − 1] AND [refineX(c_nb) ≡ false] then {recursive refinement}
4:       return false
5: {refine cells to maintain a maximum l_i difference of 1 between Cells }
6: for all neighbouring Cells c_nb ∈ Faces {W,E} do
7:    if [c_nb → l_i < l_i] AND [refineX(c_nb) ≡ false] then {recursive refinement}
8:       return false
9: Create new cell c_○, while c becomes the refined cell to the West
10: c→ {l_i = l_i + 1}
11: c→ {i = 2i}
12: c_○ → l_i = c→ l_i
13: c_○ → i = c→ i + 1
14: Add faces and modify geometric data according to Figure 7
15: Interpolate cell-centred φ using limited ∇φ
16: set c→ wasXRefined = true
17: set c_○ → wasXRefined = true
18: return true
```

Algorithm 3.3 Flag and coarsen two `Cells` in the x-direction

```
Require: all coarsening flags set to false
1: for all Cells c do
2:    Calculate least-squares c→ ∇φ
3:    Calculate c→ Δx_target
4: for all Faces f do
5:    if [f -L-> c → Δx_target ≥ 2Δx_L] OR [f -R-> c → Δx_target ≥ 2Δx_R] then
6:       f → doCoarsen = true
7:       f -L-> c → flaggedCoarsenX = true
8:       f -R-> c → flaggedCoarsenX = true
9: {go through Faces and coarsen level by level}
10: for i = l_i,min + 1 to l_i,max do
11:    for all Faces f do
12:       if [f → doCoarsen ≡ true] AND [f → wasXCoarsened ≡ false] then
13:          call coarsenX(f, i) {see algorithm 3.4}
```

Another difference exists between the present and original versions of algorithms 3.2 and 3.4; the present versions contain additional restrictions on the refinement levels allowed between neighbouring cells. Specifically, lines 5–8 of refineX() and lines 13–19 of coarsenX() specify that East and West neighbouring cells cannot be smaller than half the width of the centre cell. This limit on the change in anisotropy over a given area yields a smoother solution under compressible flow, possibly because of the simulation's sensitivity to the gradient reconstruction. On a highly anisotropic mesh consisting of high aspect-ratio cells, it is difficult to achieve a smooth and accurate reconstruction using the least-squares method.

Algorithm 3.4 coarsenX(`Face f`, level l)

1: {*make sure we are coarsening cells at the correct refinement level*}
2: **if** $\left[\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow l_i \neq l\right]$ AND $\left[\texttt{f} \xrightarrow{R} \texttt{c} \rightarrow l_i \neq l\right]$ **then**
3: return *false*
4: {*don't coarsen cells that are/were flagged for refinement*}
5: **if** $\left[\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow \text{flaggedRefineX} = \textit{true}\right]$ OR $\left[\texttt{f} \xrightarrow{R} \texttt{c} \rightarrow \text{flaggedRefineX} = \textit{true}\right]$ **then**
6: return *false*
7: {*only cells of the same dimensions can be coarsened*}
8: **if** $\left[\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow l_i \neq \texttt{f} \xrightarrow{R} \texttt{c} \rightarrow l_i\right]$ OR $\left[\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow l_j \neq \texttt{f} \xrightarrow{R} \texttt{c} \rightarrow l_j\right]$ **then**
9: return *false*
10: {*prevent brick-wall coarsening shown in Figure 5*}
11: **if** $\left[\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow i \div 2 \neq \texttt{f} \xrightarrow{R} \texttt{c} \rightarrow i \div 2\right]$ **then** {*the* $\div$ *sign denotes integer division; e.g.* $3 \div 2 = 1$}
12: return *false*
13: {*do not coarsen if the neighbouring cells are finer*}
14: **for all** `Faces` $\texttt{f} \xrightarrow{L} \texttt{c} \xrightarrow{W} \texttt{f}$ **do**
15: **if** $\left[\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow l_i > l\right]$ **then**
16: return *false*
17: **for all** `Faces` $\texttt{f} \xrightarrow{R} \texttt{c} \xrightarrow{E} \texttt{f}$ **do**
18: **if** $\left[\texttt{f} \xrightarrow{R} \texttt{c} \rightarrow l_i > l\right]$ **then**
19: return *false*
20: {*enforce the limit of* $\leqslant$ *2 neighbouring cells*}
21: **for all** `Cells` $\texttt{f} \xrightarrow{R,L} \texttt{c}$ **do**
22: **if** [`c` has $>$ 1 `Face f` $\in$ `Faces` $\{N,S\}$] **then**
23: return *false*
24: *Modify face connectivity and geometry according to Figure 8*
25: *Modify cell geometry for* $\texttt{f} \xrightarrow{W} \texttt{c}$*; average cell-centred* **u** *values*
26: $\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow \{l_i = l_i - 1\}$
27: $\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow \{i = i \div 2\}$
28: *Remove* $\texttt{f} \xrightarrow{R} \texttt{c}$ *from the list of* `Cells`
29: *Remove* `f` *from the list of* `Faces`
30: return *true*

Performing a sweep of the entire mesh to coarsen and refine the cells is not necessary at every time step. The present code operates at low CFL numbers (in the range of 0.1–0.2), which implies that the solution requires 5–10 time steps to change significantly within a cell. Because of this, the refinement algorithm is run only every 5 time steps, and the coarsening algorithm every 20. Coarsening regularly does not serve to preserve solution accuracy and is computationally more intensive than refinement, hence its relatively infrequent application. Infrequent coarsening was also found to yield overall memory savings similar to those

yielded by frequent coarsening; this was a promising result since it implied that cells were not alternating between coarsening and refinement.

3.4. Adaptation criteria

An adaptation criterion uses a function that is sensitive to some error measure or flow features to specify which cells need to be refined and coarsened. Often, this function requires the user to supply parameters for the refinement process to proceed correctly. This either allows finer control of the solution quality, or else it complicates the problem by introducing several unknowns that CFD practitioners must 'feel' their way around.

There are many adaptation criteria in the literature; however, most fall under one of the following categories:

1. *Manually-tuned gradient sensors*: For example, the criterion used for steady compressible flow by Ripley [10], based on an implementation by Lien [14]. This criterion flags cells for refinement based on the absolute value of the difference between the density in the centre cell and its neighbours. It is a simple criterion that is easy to implement and consumes little computational time. However, it is not sensitive to changes in the second derivative, and requires the user to specify a tolerance for the gradient, which depends on the flow being simulated.
2. *Combination gradient/higher order derivative sensors*: In Reference [3], Sun uses a criterion which is based on the ratio of the second-order Taylor series truncation error term to the first-order one, which, when discretized, approximates a sensor function $f(\phi)$ based on the gradients ϕ_x at the cell centre and a neighbouring face:

$$f(\phi) = \frac{|\phi_{x,\text{face}} - \phi_{x,\circ}|}{\alpha\phi_\circ/\Delta x + |\phi_{x,\circ}|} \approx \left|\frac{\phi_{xx}\Delta x}{2\phi_x}\right| \tag{7}$$

   where α is a small user-defined parameter used to prevent division by zero in the denominator. Sun used central finite differences to estimate the gradients at cell faces, and a least-squares technique to reconstruct the cell-centred gradient. Flagging cells for refinement and coarsening involves comparing the sensor function to user-specified upper and lower bounds on the ratio, ε_r and ε_c:

$$\text{if } f(\phi) > \varepsilon_\text{r} \Rightarrow \text{refine cells on either side of face}$$

$$\text{if } f(\phi) < \varepsilon_\text{c} \Rightarrow \text{join cells on either side of face}$$

   This approach yielded good results for Sun's isotropically adapted mesh, but the user-supplied parameters were flow-dependent, and the method did not perform as well on the present anisotropic mesh, due to the noisy nature of the second derivative.
3. *Optimal cell dimensions based on error criteria*: For example, consider the lowest-order term of the Taylor series truncation error (which is dependent on the order of the scheme used) integrated over the cell area, as presented in Reference [4]. For a second-order accurate scheme, this error tolerance is a function of the cell dimensions Δx, Δy, and the second derivatives $\nabla^2\phi$. Ham *et al.*, in Reference [4] obtained quasi-optimal expressions for $\{\Delta x, \Delta y\}_\text{target}$, and used those to determine the required levels of coarsening and refinement in the adaptation algorithm.

In terms of the amount of user knowledge and intervention required to operate the CFD code, method 3 is the most tractable since no assumptions about the solution need to be made. The L_∞ error norm provides a worst-case estimate of the error bound, and while there is no guarantee that this matches the actual error in the solution, it is the best that can be achieved using a Taylor remainder approximation.

For the present code, method 3 was adopted. However, instead of using the second-order term in the Taylor series, the first-order term was used to generate optimal $\{\Delta x, \Delta y\}_{\text{target}}$. Two arguments can be used to justify this choice of error indicator:

1. Although the numerical method used is formally second-order accurate, the use of gradient limiters and a mixture of quasi-first-order accurate corrections (seen in Section 3.5) reduces the code to first-order accuracy in the presence of discontinuities and sharp/oscillatory flow features.
2. The second derivative is noisy and computationally expensive to reconstruct on a non-uniform grid, which causes the corresponding adaptation to be haphazard and lose accuracy in key locations.

Using $\nabla\phi$ as the functional of interest (the present code takes $\nabla\phi = |\nabla\rho|$) and the first-order term of the Taylor remainder to be the error term, the optimal $\{\Delta x, \Delta y\}_{\text{target}}$ were found using Lagrange multipliers [15] to be:

$$\Delta x^* = \left(\frac{2\tau\phi_y}{\phi_x^2}\right)^{1/3} \tag{8}$$

$$\Delta y^* = \left(\frac{2\tau\phi_x}{\phi_y^2}\right)^{1/3} \tag{9}$$

where τ is the user-specified error tolerance.

In smooth flow regions, one or both of ϕ_x and ϕ_y may be zero; hence, to avoid division by zero in the code, practical expressions are:

$$\Delta x^* = \left(\frac{2\tau(\phi_y + \alpha)}{(\phi_x + \alpha)^2}\right)^{1/3} \tag{10}$$

$$\Delta y^* = \left(\frac{2\tau(\phi_x + \alpha)}{(\phi_y + \alpha)^2}\right)^{1/3} \tag{11}$$

where α is some small nonzero constant. In the code, $\alpha = 0.01\tau$ was chosen arbitrarily. Ideally, α should be much smaller than τ but larger than machine ε in order to avoid catastrophic cancellation.

3.5. Solution reconstruction via the gradient

As described in Section 2.4.1, the MUSCL approach is used to improve the accuracy of the numerical estimate of the cell interface flux. This involves extrapolation of the solution **u** from the centre of the cell to another point in the cell. For a formally second-order accurate

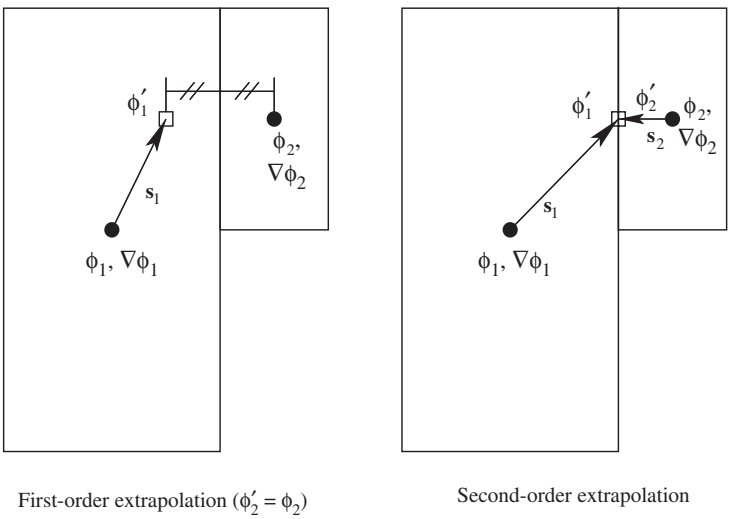

Figure 9. First- and second-order gradient extrapolation on an anisotropic Cartesian mesh: $\phi' = \phi + \nabla\phi \cdot \mathbf{s}$.

numerical method on a uniform Cartesian mesh, this extrapolation takes **u** to the face of the cell. For the corresponding first-order accurate method, no extrapolation is performed; interface fluxes are based on cell-centred **u**.

On the adapted Cartesian meshes presented in this paper, using cell-centred **u** to calculate interface fluxes results in severe degradation of the solution quality. Only by using gradient corrections is the solution rendered reasonably smooth as compared to uniform mesh results. However, for difficult test cases involving high Mach numbers and rapid expansions, even gradient-limited second-order solutions can yield negative pressures and densities—for this reason, it is sometimes necessary to have a first-order scheme that will work because of its dissipative nature.

A quasi-first-order accurate scheme can be constructed by using an auxiliary node approach: on both sides of a given interface (where the flux must be calculated), cell-centred **u** are extrapolated to points equidistant from the interface. These points lie on a line that is perpendicular to the interface and crosses its centre. The distance of the points from the interface is the minimum of the perpendicular distances between each of the cell centres and the interface. By moving the locations of the extrapolated **u** toward the interface, the scheme is extended from first to second-order accuracy. Figure 9 demonstrates the extrapolation graphically.

The least-squares method was used to estimate solution gradients. In the specific case of a uniform mesh, it can be shown that the least-squares method reduces to approximating the gradient via two-point central differences.

3.5.1. Application of gradient limiters. In addition to using $\nabla\phi$ to increase the order of accuracy of the method, the code requires gradient information during the refinement procedure to determine new cell-centred ϕ values (see Figure 7). As described in Section 2.4.2, the use of unlimited gradient information can result in negative pressure and density in some cells,

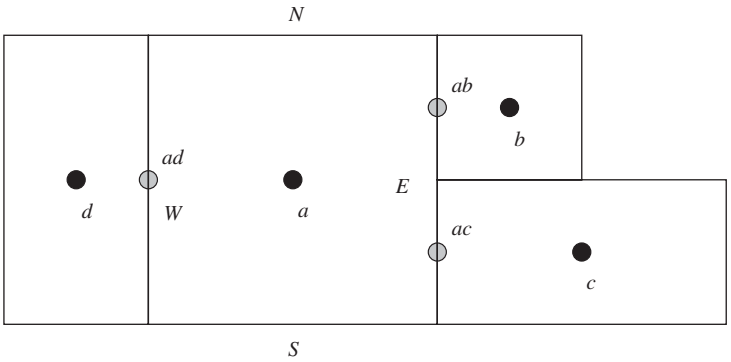

Figure 10. Mesh used for gradient limiter application in the x-direction.

which dooms the entire flow simulation. The implementation in code of the minmod and superbee limiters, as they limit x-gradients, is as follows:

Using Figure 10 as a reference, both limiters must decide which [combination] of the following gradients to use as the cell-centred value:

$$\left.\frac{\partial\phi}{\partial x}\right|_{ab}, \quad \left.\frac{\partial\phi}{\partial x}\right|_{ac}, \quad \left.\frac{\partial\phi}{\partial x}\right|_{ad} \tag{12}$$

The face gradients are approximated as in Sun's implementation [3]:

$$\left.\frac{\partial\phi}{\partial x}\right|_{ab} \approx \frac{\phi_b - \phi_a}{(\Delta x_a/2) + (\Delta x_b/2)} \tag{13}$$

For each cardinal direction, $\{\mathrm{N}, \mathrm{S}, \mathrm{E}, \mathrm{W}\}$, if the cell has two faces in a single direction, the face whose gradient has the lowest absolute value is chosen for the ensuing minmod or superbee function evaluation. For example, for the East faces of cell a in Figure 10:

$$\left.\frac{\partial\phi}{\partial x}\right|_{E} = \text{minmod}\left(\left.\frac{\partial\phi}{\partial x}\right|_{ab}, \left.\frac{\partial\phi}{\partial x}\right|_{ac}\right) \tag{14}$$

The chosen limiter function is then applied as described in Section 2.4.2, using the ratio of the $\{\mathrm{E}, \mathrm{W}\}$ pair of gradients as input arguments.

4. RESULTS FOR ANISOTROPICALLY ADAPTED MESH

The test cases used for the validation of the adaptation technique are shock diffraction over a backward step and supersonic flow over a forward step. The backward step has been studied experimentally and computationally in the literature by authors such as Skews [16], Bazhenova *et al.* [17], and Hillier [18]. The forward step has been studied computationally by van Leer [19] and Woodward and Colella [20]. Both cases are transient and therefore test the ability of the mesh to preserve the solution quality while refining and coarsening in time. Since

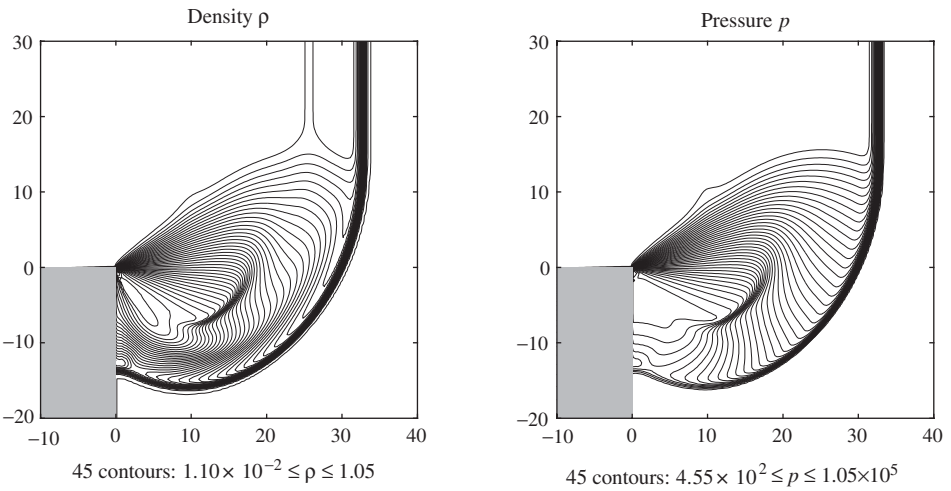

Figure 11. Mach 5.09 shock diffraction, uniform mesh: pressure and density contours.

shock diffraction over a backward step is self-similar, meshes from intermediate stages of the computation are not shown.

4.1. Backward step

The code was successfully validated against existing results for shocks with Mach numbers of 1.3, 2.4, and 5.09; only the Mach 5.09 results are shown here. The results of Figure 11 were computed on a uniform mesh with cells of dimensions $\Delta x = \Delta y = 40 \div 2^7 = 0.15625$. Adapted mesh results are shown in Figure 12, whose minimum cell dimensions are also $\Delta x = \Delta y = 0.15625$.

The mesh was adapted according to the target cell sizes defined in Equation (10) using $\tau = 10^{-4}$ and the x- and y-components of the absolute value of the gradient of density: $\nabla\phi \cdot \mathbf{e}_i = |\nabla\rho \cdot \mathbf{e}_i|$. Density was chosen as the sensor variable because it is the conservative variable that best indicates changes across shock, contact and expansion waves.

The contours of target cell size are arranged in a geometric sequence as indicated by the caption (cell area $\Delta x \Delta y$ varies geometrically with refinement level n). This information was obtained from the uniform mesh results via the calculation of $(\Delta x^*, \Delta y^*)$ for every cell. Since the refinement algorithm is relatively conservative (it refines some extra cells in order to maintain the smoothness of the mesh), the actual refined mesh has small cells in places that the $(\Delta x^*, \Delta y^*)$ information does not predict. In general, however, the qualitative agreement between the actual refined mesh and the predicted cell sizes is good.

4.2. Mach 3.0 flow over a forward step

This test case was performed with the same geometry and initial conditions that were used by Woodward and Colella [20], on an adaptive mesh with minimum possible cell dimensions $\Delta x = \Delta y = 0.2 \div 2^5 = 6.25 \times 10^{-3}$. The adaptive mesh simulation encounters negative $\{\rho, p\}$

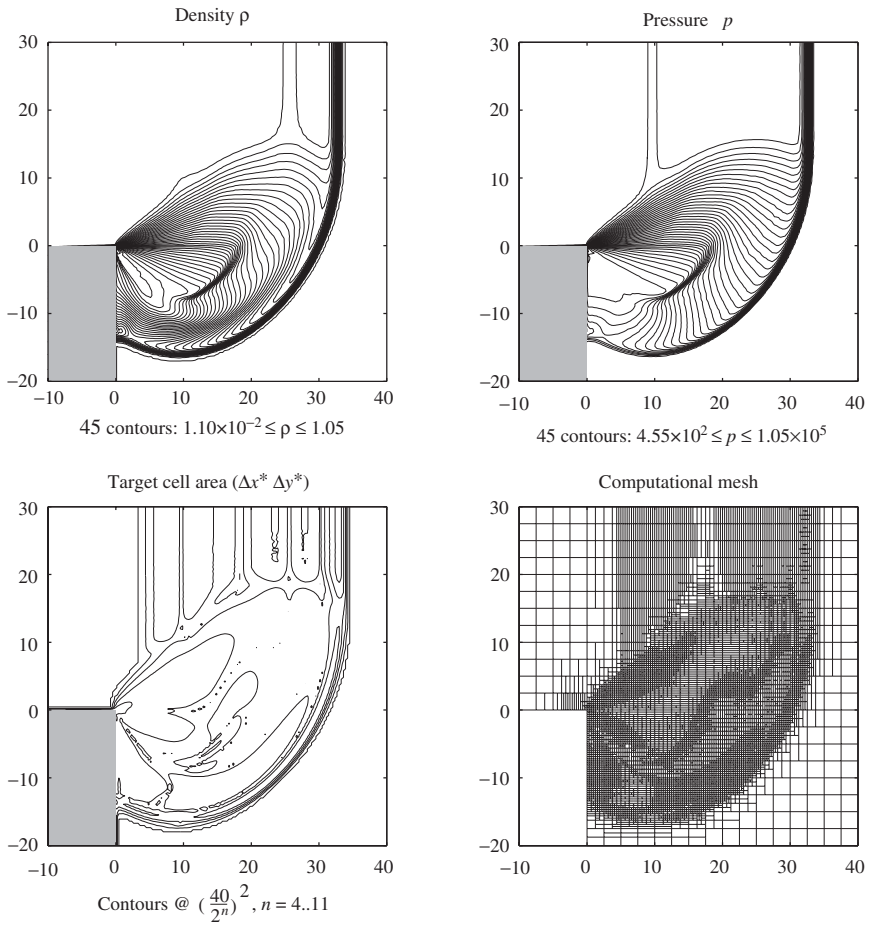

Figure 12. Mach 5.09 shock diffraction, adapted mesh: pressure and density contours.

if cells at a refinement level higher than $l_{\max} = 5$ are used. If a manual entropy fix were to be implemented at the step corner as in Reference [20], this would not be the case.

As in the backward step test case, the mesh was adapted using density as the indicator variable for obtaining the target cell sizes (defined in Equations (10) and (11)). The stricter value of $\tau = 10^{-5}$ was chosen for the error criterion because of the higher resolution of this simulation over the backward step.

Figure 13 shows the computational mesh at different points in time. Most of the shock wave structure is set up in the first two time units of the simulation; afterwards, the shock waves move relatively slowly toward their position at the end of the simulation.

Density contours are shown in Figure 14, where they are directly compared to those obtained using the uniform mesh. The density contours are spaced linearly, whereas the difference contours $\{\Delta\rho, \Delta p\}$ are spaced logarithmically.

It is apparent from Figure 14 that the uniform and adapted mesh solutions differ by at most $\{\Delta\rho, \Delta p\} = 1 \times 10^{-1}$ in the majority of the regions outside the shock waves. Although

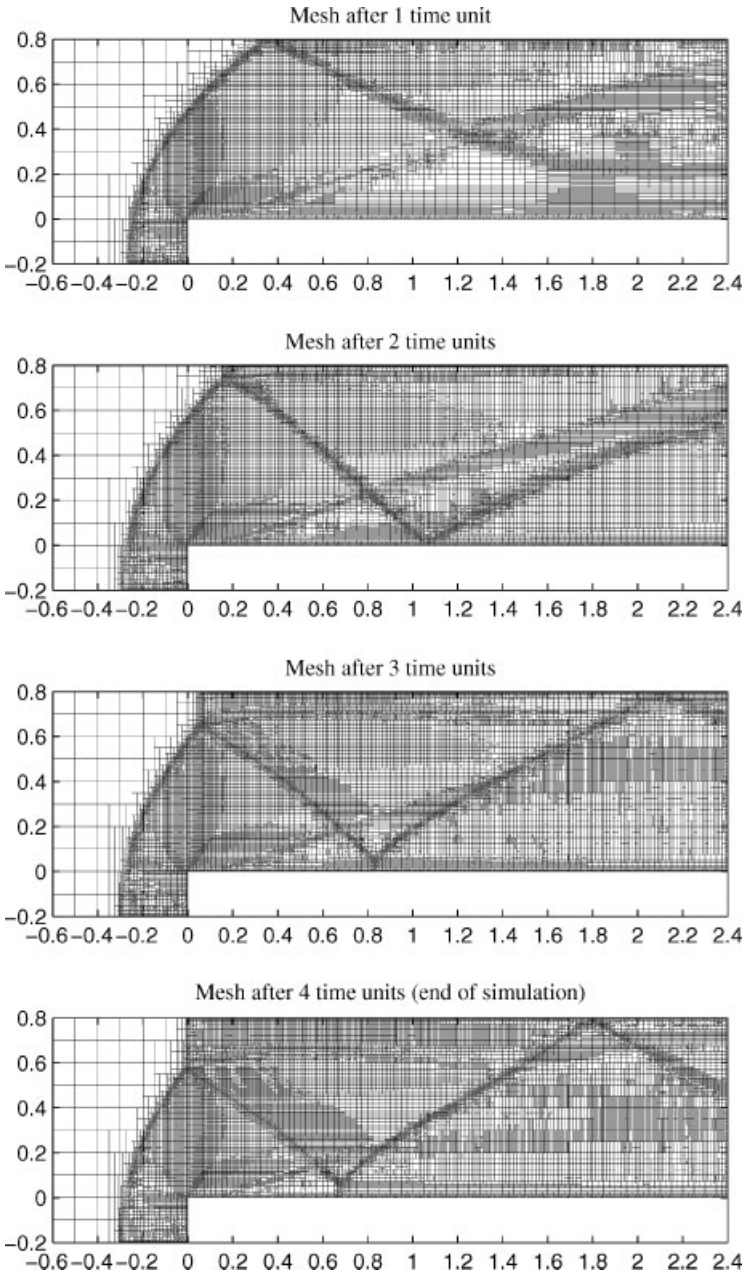

Figure 13. Mach 3.0 flow over a forward step: computational mesh.

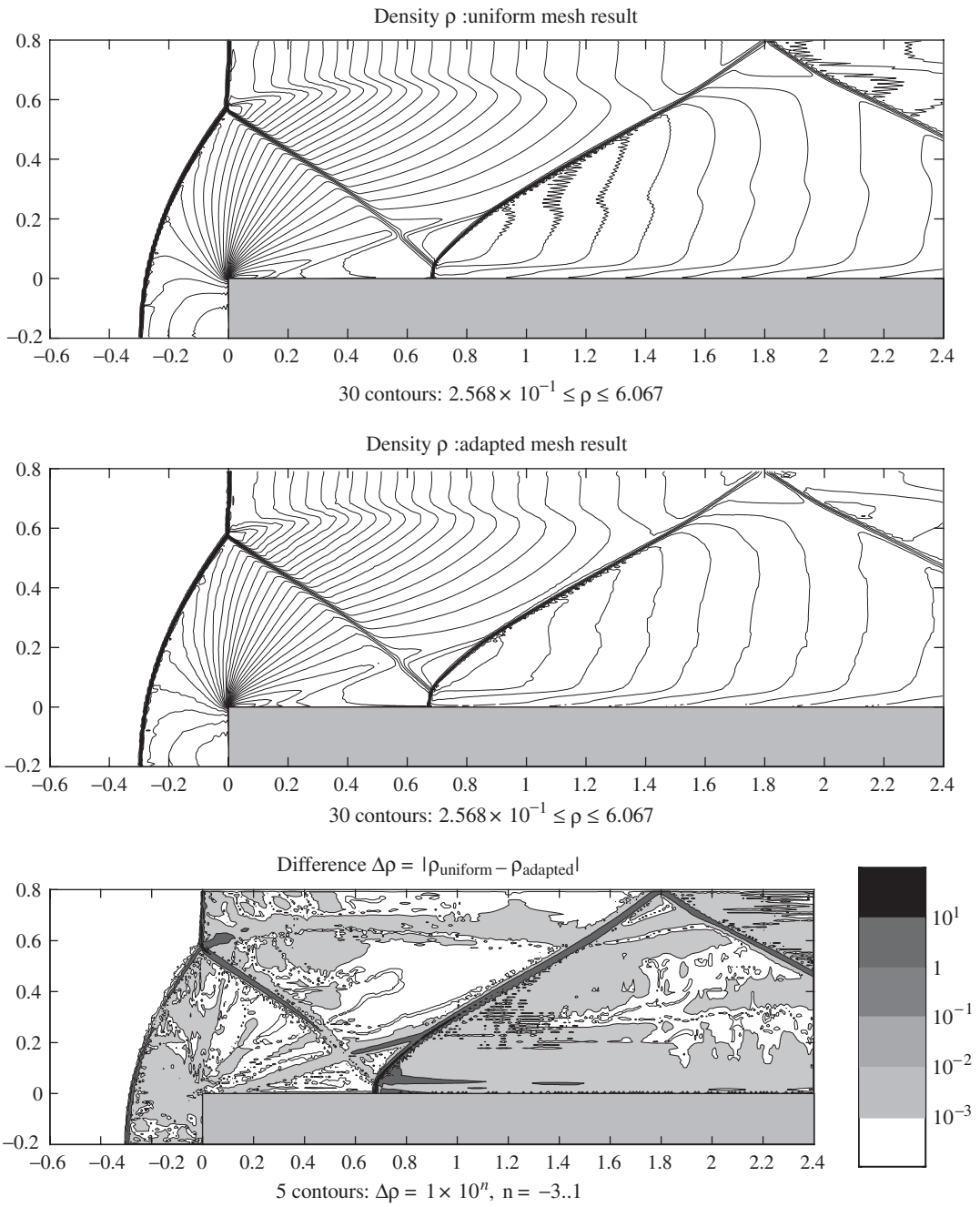

Figure 14. Mach 3.0 flow over a forward step: comparison of density field between uniform and adapted mesh results. The minimum cell dimension is the same for both the uniform and adapted meshes.

the contours within the shock regions are very tightly packed and correspond closely between the uniform and adapted mesh results, any slight perturbation to one side or the other may be responsible for causing the comparatively large solution differences on the order of $\Delta\rho \approx 10$.

As an indication of the sensitivity of this test case to the mesh adaptation, observe that the location of the shock reflection at the top of the domain (at $x \approx 1.8$) is slightly different between the uniform and adapted mesh results. At this point, however, the shock is relatively weak and therefore so is the solution difference.

One interesting implication of the results of Figure 14 is the fact that there are large areas of the solution where the differences $\Delta\rho$ exceed $\tau = 10^{-5}$. This is because the 'optimal' target cell sizes optimize cell size based on only one single, relatively simple indicator of the solution error, whereas in reality the composition of the solution error is more complicated than the gradient term of the Taylor series expansion. It is already apparent from its oscillatory behaviour and its difficulty with the step corner expansion that the uniform mesh result is non-ideal. In addition, the very act of anisotropic mesh refinement introduces errors into the solution through averaging and gradient extrapolation—errors that are partially indicated by rougher density and pressure contours.

5. CONCLUDING REMARKS

5.1. Computational resource savings

At the beginning of this paper it was promised that the anisotropic mesh adaptation technique would yield computational resource savings over uniform and equivalent isotropically adapted meshes. Since the forward step test case generally consumes the most resources, and its major flow features lie at angles offset from the cardinal directions, it was chosen as a worst-case scenario for the calculation of resource savings. Figure 15 quantifies the time and memory savings achieved over a uniform mesh, while Figure 16 quantifies the memory savings achieved over an equivalent isotropically adapted mesh. Processor time savings were tabulated and plotted according to the following expression:

$$\frac{P_{\text{anisotropic}}}{P_{\text{uniform}}} = \frac{\Delta t_{n,\,\text{anisotropic}}}{\Delta t_{n,\,\text{uniform}}} \tag{15}$$

where Δt_n is the real-world time required to reach time step n within the simulation. Note that there is no valid execution time data at time zero. Memory savings were calculated using a similar expression:

$$\frac{M_{\text{anisotropic}}}{M_{\text{uniform}}} = \frac{m_{n,\,\text{anisotropic}}}{m_{n,\,\text{uniform}}} \tag{16}$$

where $m_n = (\text{no. faces} \times m_{\text{face}}) + (\text{no. cells} \times m_{\text{cell}})$. For the present code, $m_{\text{face}} = 236$ bytes, and $m_{\text{cell}} = 352$ bytes. These quantities represent the amount of memory required by `Face` and `Cell` objects, and depend on what extra storage is used for convenience within those classes. Obviously, the memory savings would change if different values were used for m_{cell} and m_{face}.

One interesting feature of Figure 15 is the fact that the processing time savings are not proportional to the memory savings. For a uniform mesh, the required processing time is

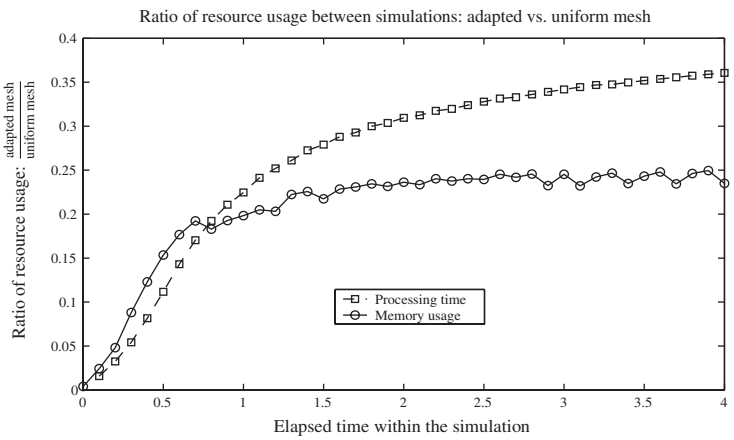

Figure 15. Memory and CPU time savings provided by anisotropic mesh adaptation for the forward step test case.

directly proportional to the memory used, since the same number of operations must be performed on each `Cell` and `Face` object. However, there are two key differences between adaptive and uniform meshes that explain the difference in the processing time to memory usage ratio:

1. The ratio of the number of cells to the number of faces is not necessarily the same for adapted and uniform meshes. For the forward step test cases, the ratios were found to be

$$\left(\frac{\text{no. faces}}{\text{no. cells}}\right)_{\text{uniform}} \approx 2.007, \quad \left(\frac{\text{no. faces}}{\text{no. cells}}\right)_{\text{adapted}} \approx 2.165 \tag{17}$$

   Without going into the details of the code implementation, this discrepancy between `Cell`/`Face` ratios affect the computational time since different operations are performed on `Cells` and `Faces`.
2. The act of mesh refinement generally increases the amount of processor time required—the uniform mesh obviously does not require this functionality.

For the memory usage comparison of Figure 16, the isotropic mesh memory usage was calculated based on the assumption that all of the anisotropic cells in the mesh would be subdivided in the appropriate direction (either x- or y-refined) until their aspect ratios were equal to the equivalent isotropically refined cells. This assumption is not strictly correct, since subdividing anisotropic cells with high aspect ratio would not necessarily yield exactly the equivalent isotropic mesh. However, given the conservative nature of the coarsening algorithm and the fact that the refinement criterion is based on flow gradients, the approximation seems reasonable. Surprisingly for a flow whose major features lie at an angle to both of the cardinal directions, the anisotropic mesh offers significant memory savings over the equivalent isotropic mesh.

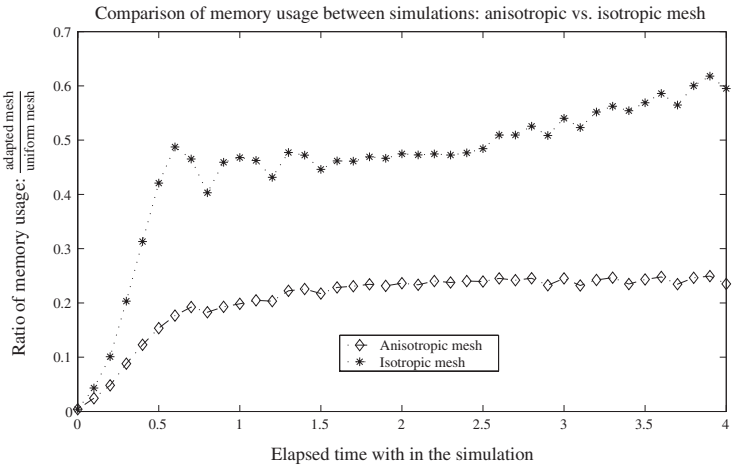

Figure 16. Memory savings provided by anisotropic adaptation for the forward step test case.

Table III. Division of processor time among computational tasks: forward step test case.

	Code	
Task	Uniform (%)	Adaptive (%)
Refine cells	0	23
Coarsen cells	0	2
Calculate AUSM$^+$ fluxes	25	15
Step forward in time	19	26
Calculate $\{\rho, p, \mathbf{u}\}$ from $\mathbf{u}$	3	2
Other tasks	53	32

Although the anisotropic code has the ability to refine isotropically, it is not instructive to compare the processor time required for the anisotropic vs. isotropic meshes. The isotropic mesh data structures defined in Reference [2] are operated upon differently than the list used in the present code, and have even been vectorized (see Reference [3]) with the probable result that they are computationally more efficient. It is difficult, however, to quantify this efficiency without setting up a test case and running the code on the same platform that was used for the present anisotropic code.

Table III outlines the percentage of processor time spent on the main computational tasks. The category 'other tasks' is primarily composed of: (a) applying the gradient limiter; (b) calculating cell-centred and face-based gradients; (c) solution file output; however, some of these tasks are used by the refinement and coarsening procedures, and so to avoid 'double-accounting', only the major tasks are shown. It is interesting to note that the adaptive simulation consumes more processing time when stepping forward in time. Code execution time profiles indicate that this is due to the requirement that the each `Cell` search lists of neighbouring `Face` objects in order to add their flux contribution.

5.2. Future direction

In order for the anisotropic Cartesian adaptation technique to be effective for a wider range of supersonic flows and geometries, the areas outlined below should be examined in further detail and implemented in code wherever possible.

Cut-cell method: This straightforward improvement has already been implemented in similar codes by de Zeeuw and Powell [2] and Sun [3]. The cut-cell method allows for improved resolution of arbitrary boundaries by approximating them with a line (or plane) that 'cuts' the computational cells. Boundary conditions are implemented by setting velocity normal to the 'cut' equal to zero and interpolating the boundary pressure in the same way as described in Section 2.2.1.

Extension to three dimensions: The data structure and algorithms were extended by Ham *et al.*, in Reference [4] to three dimensions; therefore, extension of the present code is technically feasible. In three dimensions, however, keeping track of adjacent faces in the code becomes complicated, and therefore it is worth examining the present code for possible simplifications before taking on this task.

Solution quality improvement: It is well documented that waves travelling between cells of different aspect ratios experience distortion [11, 5] and so the algorithms in this paper are stricter than the original ones of Ham *et al.*, in that they place greater limits on the differences in aspect ratio between adjacent cells. Wu and Li [5] have proposed that such differences can adversely affect the level of numerical dissipation present, which would suggest that the numerical methods should take neighbouring cells' aspect ratios into account when determining fluxes.

Improvement in the quality of the solution would also serve to provide a better estimate of higher-order derivatives, opening up the possibility of using them in alternate mesh refinement criteria.

APPENDIX A: DERIVATION OF THE OPTIMAL CELL SIZE CRITERION

Consider the Taylor series expansion of $\phi(\mathbf{x})$ about $\mathbf{x}_\circ$, the cell centre:

$$\phi(\mathbf{x}) = \phi(\mathbf{x}_\circ) + [\nabla\phi(\mathbf{x})]^{\mathrm{T}}\mathbf{x} + \tfrac{1}{2}\,\mathbf{x}^{\mathrm{T}}[\nabla^2\phi(\mathbf{x})]\mathbf{x} + O(\mathbf{x}^3) \tag{A1}$$

For a first-order accurate numerical scheme, the leading error term is $[\nabla\phi(\mathbf{x})]^{\mathrm{T}}\mathbf{x}$. By assuming that the maximum absolute error occurs at the farthest point from the cell centre, in two dimensions the expression becomes $\frac{\Delta x}{2}\phi_x + \frac{\Delta y}{2}\phi_y$.

Integrating this term over the cell provides an expression for the L_∞ error:

$$\varepsilon_{\phi,L_\infty} = \int_{-\frac{\Delta x}{2}}^{\frac{\Delta x}{2}} \int_{-\frac{\Delta y}{2}}^{\frac{\Delta y}{2}} \frac{\Delta x}{2}\,\phi_x + \frac{\Delta y}{2}\,\phi_y \,\mathrm{d}y\,\mathrm{d}x = \frac{\Delta x^2 \Delta y}{4}\,\phi_x + \frac{\Delta x \Delta y^2}{4}\,\phi_y \tag{A2}$$

We are now in a position to formulate the optimization problem:

$$\text{maximize} \quad \Delta x \Delta y \tag{A3}$$

$$\text{subject to} \quad \varepsilon_{\phi,L_\infty} \leqslant \tau \tag{A4}$$

$$\Delta x \geqslant 0 \tag{A5}$$

$$\Delta y \geqslant 0 \tag{A6}$$

where τ is a user-specified error tolerance. For ϕ a single variable such as density, this problem can be solved analytically using Lagrange multipliers. Following the theory of optimization using Lagrange multipliers as outlined in Reference [15], we formulate and minimize the following Lagrange function:

$$\mathscr{L}(\Delta x, \Delta y, \lambda) = -\Delta x \Delta y - \lambda \left(\tau - \frac{\Delta x^2 \Delta y}{4} \phi_x - \frac{\Delta x \Delta y^2}{4} \phi_y \right) \tag{A7}$$

Setting the derivative of $\mathscr{L}$ with respect to $\mathbf{x} = [\Delta x, \Delta y]^{\mathrm{T}}$ equal to zero yields the system of equations:

$$\nabla_{\mathbf{x}} \mathscr{L} = \begin{bmatrix} -\Delta y^* + \lambda^* \dfrac{\Delta x^* \Delta y^*}{2} \phi_x + \lambda^* \dfrac{\Delta y^{*2}}{4} \phi_y \\ -\Delta x^* + \lambda^* \dfrac{\Delta x^{*2}}{4} \phi_x + \lambda^* \dfrac{\Delta x^* \Delta y^*}{2} \phi_y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{A8}$$

where Δx^*, Δy^* and λ^* are optimal variable values. This system can be reduced to

$$\lambda^* \Delta x^* \phi_x = \lambda^* \Delta y^* \phi_y \tag{A9}$$

It is safe to assume that the *strict complementarity* condition applies; in other words, constraint (A4) is active (equality holds) at the optimal solution, $\mathbf{x} = [\Delta x^*, \Delta y^*]^{\mathrm{T}}$; thus, $\lambda^* \neq 0$ and it can be divided out of the equation. The remaining expression,

$$\Delta x^* \phi_x = \Delta y^* \phi_y \tag{A10}$$

provides the value of Δx^* in terms of Δy^*; by using this in the *equality* constraint corresponding to (A4), we obtain the following expressions for the optimal target cell dimensions:

$$\Delta x^* = \left(\frac{2\tau \phi_y}{\phi_x^2} \right)^{1/3} \tag{A11}$$

$$\Delta y^* = \left(\frac{2\tau \phi_x}{\phi_y^2} \right)^{1/3} \tag{A12}$$

ACKNOWLEDGEMENTS

The authors are grateful to the Natural Sciences and Engineering Research Council of Canada (NSERC) for the financial support of this work, and to Dr Meng-Sing Liou of the NASA Glenn Research Center for his helpful discussions about the AUSM scheme.

REFERENCES

1. Leveque RJ. *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press: Cambridge, 2002.
2. De Zeeuw D, Powell KG. An adaptively refined Cartesian mesh solver for the Euler equations. *Journal of Computational Physics* 1993; **104**:56–68.
3. Sun M. Numerical and experimental studies of shock wave interaction with bodies. *Ph.D. Thesis*, Tohoku University, 1998.
4. Ham F, Lien FS, Strong AB. A Cartesian grid method with transient anisotropic adaptation. *Journal of Computational Physics* 2002; **179**:469–494.
5. Wu Z-N, Li K. Anisotropic Cartesian grid method for steady inviscid shocked flow computation. *International Journal for Numerical Methods in Fluids* 2003; **41**:1053–1084.
6. Laney CB. *Computational Gasdynamics*. Cambridge University Press: Cambridge, 1998.
7. Liou M-S. A sequel to AUSM: AUSM$^+$. *Journal of Computational Physics* 1996; **129**:364–382.
8. Liou M-S, Steffen CJ. A new flux splitting scheme. *Journal of Computational Physics* 1993; **107**:23–29.
9. Meinke M, Schröder W, Krause E, Rister Th. A comparison of second- and sixth-order methods for large-eddy simulations. *Computers and Fluids* 2001; **31**:695–718.
10. Ripley RC. Unstructured-grid methods for numerical simulation of shock wave interactions with bodies and boundary layers. *Master's Thesis*, University of Waterloo, 2002.
11. Larsson NJ. Computational aero acoustics for vehicle applications. *Licentiate Thesis*, Chalmers University of Technology, 2002.
12. Toro EF. *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer: Berlin, 1999.
13. Lee RC, Tepfenhart WM. *UML and C++: A Practical Guide to Object-Oriented Software Development*. Prentice-Hall: Upper Saddle River, 1997.
14. Lien F-S. A pressure-based unstructured-grid method for all-speed flows. *International Journal for Numerical Methods in Fluids* 2000; **33**:355–374.
15. Nocedal J, Wright SJ. *Numerical Optimization*. Springer: New York, 1999.
16. Skews BW. The perturbed region behind a diffracting shock wave. *Journal of Fluid Mechanics* 1967; **29**(4):705–719.
17. Bazhenova TV, Gvozdeva LG, Nettleton MA. Unsteady interactions of shock waves. *Progress in Aerospace Science* 1984; **21**:249–331.
18. Hillier R. Computation of shock wave diffraction at a ninety degrees convex edge. *Shock Waves* 1991; **1**:89–98.
19. van Leer B. Towards the ultimate conservative difference scheme. v. a second-order sequel to Godunov's method. *Journal of Computational Physics* 1979; **32**:101–136.
20. Woodward P, Colella P. The numerical simulation of two-dimensional fluid flow with strong shocks. *Journal of Computational Physics* 1984; **54**:115–173.

