INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 22, 85–101 (1995)

ADAPTIVE SOLUTIONS FOR UNSTEADY LAMINAR FLOWS ON UNSTRUCTURED GRIDS

R. VILSMEIER AND D. HÄNEL

Institut für Verbrennung und Gasdynamik, University of Duisburg, D-47048 Duisburg, Germany

SUMMARY

An adaptive finite volume method for the simulation of time-dependent, viscous flow is presented. The Navier–Stokes equations are discretized by central schemes on unstructured grids and solved by an explicit Runge–Kutta method. The essential topics of the present study are a new concept for a local Runge–Kutta time-stepping scheme, called multisequence Runge–Kutta, which reduces the severe stability restriction in unsteady problems, a common grid generation and adaptation procedure and the application of dynamic grids for capturing moving flow structures. Results are presented for laminar, separated flow around an aerofoil with a flap.

KEY WORDS: Navier–Stokes equations; time-dependent, separated flow; unstructured, adaptive, dynamic grids; local time-stepping scheme

1. INTRODUCTION

Methods of solution based on unstructured grids enable a high degree of flexibility with respect to solution-adaptive grid concepts. Grid cells can be added, removed or deformed during the solution according to criteria derived from requirements of accuracy. Therefore such adaptive methods are in principle ideal methods to deal with flow problems of different characteristic scales, where a high resolution is required in parts of the integration domain while in other parts a moderate resolution is sufficient. In general the locations of high-resolution ranges are not known *a priori*. In non-adaptive methods the solution has to be estimated before arranging the grid or a sufficiently fine, global grid has to be used. Adaptive grid methods find these locations 'automatically' during the solution and adapt the grid to correspond to the actual solution. Thus the higher effort per grid point for unstructured, adaptive methods can be compensated by the sparse, effective use of grid cells.

The difficulties in formulation and application of an adaptive method depend strongly on the character of the solution. For inviscid flows, where besides geometrical features only distinct discontinuities (shocks) appear, unstructured grids are very suitable, since their geometric freedom and the ease of adapting meshes to local requirements are often more important than the advantages of structured meshes in terms of efficiency per node computed. A large number of applications can be found in the literature (see e.g. References 1–3).

Solution of the Navier–Stokes equations at high Reynolds numbers are in general much more difficult to adapt and are much less highly developed than for inviscid flows. A crucial problem is the presence of very different viscous scale lengths in different directions, e.g. body length, boundary layer thickness and vortex extensions. For adaptive methods the problem arises that often more than one adaptation criterion, different in value and direction, has to be satisfied. Another numerical difficulty is the generation of deformed cells, e.g. flat, triangular grids necessary to meet the anisotropic solution in thin shear layers. These deformed cells can introduce inaccuracies and an additional stiffness which

CCC 0271–2091/96/020085–17

Received December 1993
Revised May 1994

impairs the performance of the solution algorithms. Despite these difficulties, various authors are concerned with viscous computations on unstructured grids (see e.g. References 4–10).

A special challenge for adaptive methods is the computation of viscous, unsteady flows, which is the topic of the present paper. Many viscous flow problems, in particular at high Reynolds numbers, can become unsteady by self-induced flow separation, from which systems of moving vortices (vortex streets) develop. Typical examples are the flow around aerofoils at high angle of attack and that behind blunt bodies. The demands on an adaptive method for such flows are high. Firstly the algorithm has to be sufficiently accurate in time. Using an explicit scheme, as done here, stability restrictions lead to a severe reduction in efficiency owing to the very different sizes of grid cells. To reduce this effect in time-dependent flows, two extended versions of an explicit time-stepping Runge–Kutta scheme were developed and tested. The two versions are based on the idea of increasing the stability limit locally either by increasing the number of intermediate stages of the Runge–Kutta scheme or by applying multiple sequences of the Runge–Kutta schemes with a constant number of stages.

To utilize the advantages of adaptive algorithms for unsteady solutions, a moving, dynamic mesh should be used to adapt the details of the flow so that zones of fine resolution move with the critical flow structures. A corresponding attempt is presented here based on a combination of a static mesh superposed by an additional dynamic mesh with time-dependent grid cell distributions.

The quality of the adaptive, time-dependent algorithm is demonstrated by computational results for laminar, separated flow over an aerofoil with a flap.

2. GOVERNING EQUATIONS AND SPATIAL DISCRETIZATION

2.1. *Governing equations*

The numerical solution is based on the two-dimensional, time-dependent Euler or *Navier–Stokes* equations written in conservative integral form:

$$\int_\tau Q_t \, \mathrm{d}\tau + \oint_A (F - S)\mathrm{d}y - \oint_A (G - R)\mathrm{d}x = 0. \tag{1}$$

Here

$$Q = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix}, \quad F = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho u H_t \end{pmatrix}, \quad G = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho v H_t \end{pmatrix}, \quad S = \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ s_4 \end{pmatrix}, \quad R = \begin{pmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ r_4 \end{pmatrix},$$

where Q is the vector of conservative variables, F and G describe the inviscid flux contributions (Euler terms) and S and R are the viscous terms in a Cartesian frame (x, y, t). The abbreviated terms in the fluxes S and R are $s_4 = u\tau_{xx} + v\tau_{xy} + q_x$ and $r_4 = u\tau_{xy} + v\tau_{yy} + q_y$, with τ_{xx}, τ_{yy} and τ_{xy} the components of the stress tensor of laminar flow and q_x and q_y the components of the heat flux vector $\vec{q} = \lambda \nabla T$. The gas is assumed to be perfect.

2.2. *Spatial discretization*

A finite volume method is applied to discretize the conservation equations (1) in a mesh of triangular control volumes around a point $P(x, y)$. The discrete equations for the volume-averaged

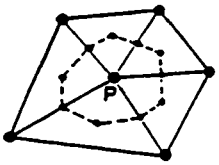

Figure 1. Control volumes for node P

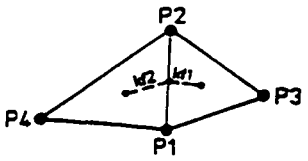

Figure 2. Basic cell for edge P1–P2

conservative variables Q on a node P read

$$\tau \left.\frac{\Delta Q}{\Delta t}\right|_P + Res_\Delta(P) = 0, \tag{2}$$

where τ is the area of the control volume and Δt is the time step. The residual $Res_\Delta(P)$ is the steady state operator, which consists of the discrete fluxes over the boundaries of the control volume.

The finite control volumes for the inviscid fluxes are defined here by a cell vertex arrangement, resulting in a central scheme of simple algorithmic structure. The boundaries of a cell vertex control volume consist of natural edges of the mesh, as sketched in Figure 1 (outer contour).

For each natural non-boundary edge the connectivity is stored in a basic cell consisting of the two nodes forming the edge as well as the opposite nodes of the two neighbouring triangles (Figure 2). In the cell vertex case the inviscid fluxes are computed as an average of neighbouring data over the edge $P1$–$P2$ and distributed to the nodes $P3$ and $P4$.

Algorithms with central discretizations require additional artificial damping terms. The damping terms are used as high-frequency filters and are computed as fourth differences D_4 of the conservative variables. This is done by first computing the second differences D_2 of variables for each node in the form of an unweighted, discrete Laplacian. The fourth differences D_4 are then computed as the second differences of the values D_2. This formulation corresponds in essence to that proposed by Mavriplis[11] and offers a cheap way of computing the damping terms. Obviously this simple formulation does not hold for a linear field in a strongly stretched mesh; however, the damping terms are consistent and small by definition.

The scheme based on central formulations of the inviscid terms has proven to be well-suited not only for the flows considered here but also for supersonic flows at moderate Mach numbers. If, however, strong shock waves are embedded, higher-order upwind schemes in conjunction with a node-centred arrangement achieve better results and are preferred for such problems in the place of central schemes.[10]

The viscous terms are approximated throughout by a node-centred formulation. The control volumes in node-centred formulations are surrounded by a set of lines from the centres of the triangles to the centres of the edges (Figure 1, broken line). First derivatives of the corresponding quantities of viscous fluxes are calculated in each triangle and projected to the boundary segments of the node-centred volume. These fluxes are distributed to the nodes $P1$ and $P2$.

2.3. Memory access

The computations were performed on an RISC workstation. Owing to the limited size of the data storage memory, it is essential to access the memory in as ordered a way as possible, allowing the computer to load packages of data for processing and thus avoiding wait states for memory requests. There exist several methods for mesh reordering. However, a compromise between optimizing memory access and the execution time for reording has to be made. Since in our case, owing to the adaptive mesh, frequent reordering is required, a simple and very fast method is used.

1. All nodes are sorted in a Cartesian direction.
2. The edges are sorted with increasing node addresses.

Although simple, the performance could be increased by a factor of two compared with unordered data access.

3. INTEGRATION IN TIME

Integration in time is carries out by an explicit time-stepping Runge–Kutta scheme. The explicit structure is well-suited for computations on unstructured grids because of its small range of coupling with neighbouring cells. The Runge–Kutta time-stepping scheme is an approved concept for solutions of the Euler and Navier–Stokes equations, in particular for steady state computations in combination with additional acceleration Techniques such as local time steps, residual smoothing and multigrid.

For time-accurate computations the efficiency of an explicit scheme can be essentially impaired by the stability restriction if the sizes of the discrete control volumes differ very strongly. Then only the smallest value of all local time steps can be used, although in large parts of the integration domain much larger steps could be chosen according to the stability limit. This situation is typical for adaptive solutions where cells are concentrated in regions of strong changes and are removed in other ranges.

To overcome these severe restrictions for unsteady computations, two improved versions of the explicit Runge–Kutta scheme were investigated and tested here. One version uses locally different numbers of intermediate stages and the other version employs locally a multiple number of Runge–Kutta sequences consisting of a constant number of intermediate stages. Both versions are based on the idea of increasing the stability limit locally, where necessary, such that for all cells a large, global time step (larger than the smallest one) can be used. In this case computational time can be saved, since additional work is only necessary for cells with smaller characteristic time steps.

This principle itself is not new.[12,13] For the Euler or Navier–Stokes equations, however, a decoupling in time between neighbouring nodes is not allowed. The reason is that the leading terms of these equations are first or second derivatives in space. The calculation remains consistent in time if an appropriate synchronization is used such that each evaluation of residuals uses quantities from neighbouring points which are on the same time level.

In the following the basic Runge–Kutta scheme will be described briefly and both extended versions will be discussed in more detail.

3.1. Basic Runge–Kutta scheme

The basic five-stage Runge–Kutta scheme reads

$$
\begin{aligned}
Q^1 &= Q^0 + \alpha_1 \cdot CFL \cdot \Delta t \cdot Res_\Delta^0, \\
Q^2 &= Q^0 + \alpha_2 \cdot CFL \cdot \Delta t \cdot Res_\Delta^1, \\
&\vdots \\
Q^{Nk} &= Q^0 + \alpha_{Nk} \cdot CFL \cdot \Delta t \cdot Res_\Delta^{Nk-1}, \\
Q^0 &= Q^{Nk},
\end{aligned}
$$

where Nk is the number of stages, α_k are the Runge–Kutta coefficients, Q^0 is the set of basic variables and Q^1 to Q^{Nk} are represented by *one* set of intermediate variables. Generally five stages are used here with coefficients α_k determined for second-order accuracy in time and maximum CFL number.

For the next two subsections the following definitions are useful.

1. The basic time step $\Delta t_{b,i}$ is the shortest time step in which characteristics starting anywhere at the cell interface for a node i reach that node i. It is a local quantity and corresponds to the time step for $CFL = 1$.
2. The minimum basic time step $\Delta t_{b,\min}$ is the lowest value of $\Delta t_{b,i}$ from all nodes.
3. Update (U) means the action of updating intermediate variables in the Runge–Kutta cycle. Evaluation (E) is the action of determining a new residual. Its prerequisite is an update. Storing (S) is the action of storing intermediate variables onto the basic variables. Its prerequisite is an update.

3.2. Runge–Kutta scheme with variable number of stages

The idea of this approach uses the fact that the largest *CFL* number of a time-stepping scheme given by a stability analysis increases with the number of intermediate stages. For a corresponding linear equation the largest *CFL* number is related to the number Nk of Runge–Kutta stages by

$$CFL_{\max,\,\mathrm{lin}} = Nk - 1. \tag{3}$$

Since the 'worst' node runs with a very large number of stages, $Nk_{\max}$, and therefore with a high *CFL* number, other more stable nodes i may perform the same time step with a lower number of stages, Nk_i. Using the theoretical values of equation (3), one obtains the required number of stages per node i as

$$Nk_i = (Nk_{\max} - 1)\frac{\Delta t_{\mathrm{b,\,min}}}{\Delta t_{\mathrm{b},i}} + 1. \tag{4}$$

As discussed above, each residual evaluation requires neighbours at the same level of time also at the intermediate levels. For the upper stages this is automatically the case if the time-accurate Runge–Kutta coefficients

$$\alpha_k = (Nk - k + 1)^{-1} \tag{5}$$

are used, where k is the stage within the cycle. From high to low these coefficients are $1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots$. A problem arises for the lower stages if two neighbours require different numbers of stages. Let j be the neighbour of i with the following condition: $Nk_i > Nk_j$. For the stages $1 \leqslant k < Nk_i - Nk_j$ the node j has to be updated, although this would not be required for its own Runge–Kutta cycle. These low-stage updates for the node j are done using the residual computed at stage 0, i.e. the beginning of the Runge–Kutta cycle. The node j is allowed to be updated with old residuals, since there is no characteristic starting at its cell interface at stage 0 that may overrun it in the first $Nk_i - Nk_j$ stages. This is in contrast with equation (4). Nodes farther away do not need to be on time if none of their neighbours requires a residual evaluation.

Grouping concept. We would probably lose all potential gain in performance if each node were treated individually. Thus a grouping concept is required. Groups are identified by the number of states, Nk_i, required by their nodes. Since the variety of basic time steps $\Delta t_{b,i}$ is large, it is also appropriate to choose a large variety of stage numbers Nk_g. In our computations we choose groups with the following numbers of stages:

$$Nk_g = 2^{ng-g}, \tag{6}$$

where g is the group index and ng is the number of groups.

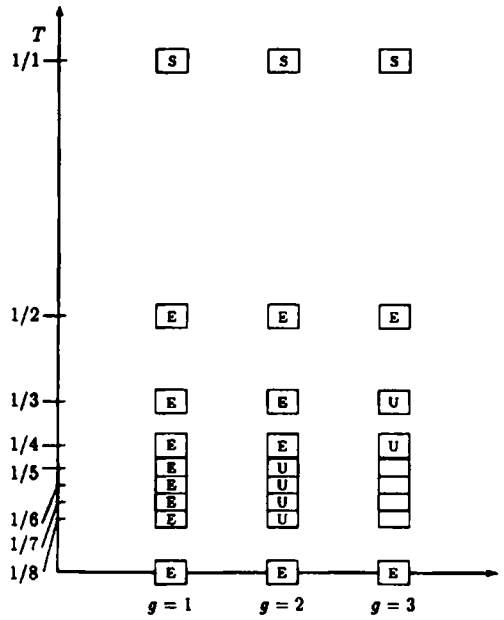

Figure 3. Evolution in time for Runge–Kutta scheme with variable number of intermediate stages. Example for three groups based on powers of two. U, update; E, evaluate residual; S, storing; $T = t/(CFL \cdot \Delta t_{b,\min})$

Note that the number of stages for groups, Equation (6), may be chosen in any other convenient monotonically decreasing order. Figure 3 shows an example for a three-group Runge–Kutta scheme based on powers of two. After analysing the limits for each node, Equation (4), they are inserted in the groups with the next higher value Nk_g, ensuring that neighbouring nodes never have a group index difference greater than one. This is done by a recursive loop over all edges as they address neighbouring nodes. The edges themselves are sorted using the same grouping concept. All edges get the lowest group index of the four nodes they support (see Figure 2).

Evaluation and update rule. Since the number of stages in the groups decreases monotonically with the group index, a residual evaluation for group g is always accomplished by an evaluation of all groups from 1 to $g - 1$. We may therefore formulate the following evaluation (E) and update (U) rule.

For the residual evaluation (E) up to group g the fluxes over all edges in groups 1 to g have to be computed. This requires a previous update (U) of all nodes in groups 1 to $min(g + 1, ng)$.

It has to be mentioned that there is a slight time delay for the fourth-order damping terms D_4, since these terms require data from up to the second neighbouring cells. The effect is negligible if the second differences D_2 (see Section 2.2) from the lower time level are stored and thus available.

Computational efficiency. Test calculations have shown that the concept of a variable number of intermediate stages performs sufficiently well, but the algorithm did not improve its computational speed as much as expected from the linear theory. The essential reason is that the *CFL* number of the non-linear system of equations does not rise with the number of stages as expected from the linear theory, Equation (3). Upon testing the basic versions of the Runge–Kutta scheme with various numbers of stages, it was found that a three-stage Runge–Kutta scheme with $CFL = 1{\cdot}2$ runs well, while a 64-stage Runge–Kutta scheme fails except with $CFL \leqslant 8{\cdot}0$. The relatively low *CFL* condition for large stage numbers reduces considerably the global advance in time. Computations with the Runge–Kutta scheme with a variable number of stages result in a speed-up factor of only around two compared with a carefully tuned five-stage basic algorithm.

3.3 *Multisequence Runge–Kutta scheme*

The discouraging results of the Runge–Kutta scheme with a large number of stages means that we probably went in the wrong direction. Experiences in many applications have shown that the Runge–Kutta scheme is most efficient in non-linear cases for a moderate number of intermediate stages. Keeping the number of stages constant now, usually between three and five, the maximum time step can be increased by employing a sequence of complete Runge–Kutta cycles locally, where the number of cycles can vary from point to point. Thus one Runge–Kutta cycle at a node i may be a fraction of a cycle of another node j.

The cycle fraction is computed in a similar way as described in the previous sub-section. Again compared with the worst condition, the cycle fraction is defined as

$$C_i = \frac{\Delta t_{\mathrm{b,min}}}{\Delta t_{\mathrm{b},i}}. \tag{7}$$

If, for example, a node i has to perform one cycle, the worst node has to perform more than one cycle, namely $1/C_i$ cycles.

Synchronization and grouping concept. To ensure accuracy in time, the sequences and intermediate stages have to be synchronized. Essential for the evaluation of the residuals is to provide updated variables at neighbouring nodes which are on the same time level.

The levels of the intermediate stages are determined by the coefficients α_k of the Runge–Kutta scheme. The following set of coefficients has allowed synchronization:

$$\alpha_k = (\tfrac{1}{2})^{Nk-k}, \tag{8}$$

where again Nk is the number of stages and k is the stage. From high to low these coefficients are $1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$. Note that second-order time accuracy is guaranteed, since the two highest coefficients are 1 and $\frac{1}{2}$, and that the set is near the set of maximum *CFL* number. With the coefficients (8) the time levels fit each other if the cycle fractions C_i are powers of $\frac{1}{2}$ (see Figure 4). Following the grouping idea mentioned in the previous subsection, groups of the same cycle fraction are introduced:

$$C_g = (\tfrac{1}{2})^{g-1}, \quad g = 1, 2, 3, \ldots. \tag{9}$$

All nodes are now analysed for their individual cycle fraction and then inserted in the group with the next higher value C_g. Again two neighbouring nodes may never have a group index difference greater than one. Edges are grouped as in the previous subsection. They get the lowest group index of the four nodes they support.

Evaluation and update rule. The evaluation and update rule is the same as formulated for the scheme with a variable number of stages. Additionally, a storing (S) is performed for all members of groups reaching the end of a cycle. A complete time step is finished when the group with the lowest cycle fraction, i.e. the group with the highest index $g = ng$, has finished one cycle. The physical time for this step is

$$\Delta t = CFL \cdot \Delta t_{\mathrm{b,min}} \cdot 2^{ng-1}, \tag{10}$$

where *CFL* is the *CFL* number for one Runge–Kutta cycle. Figure 4 shows an example for a three-group, three-stage Runge–Kutta scheme.

Computational efficiency. The multisequence Runge–Kutta scheme was applied to the problem of viscous, unsteady flow around an aerofoil, as described later. The multisequence Runge–Kutta scheme with three stages and six groups was compared with the basic five-stage scheme. After several

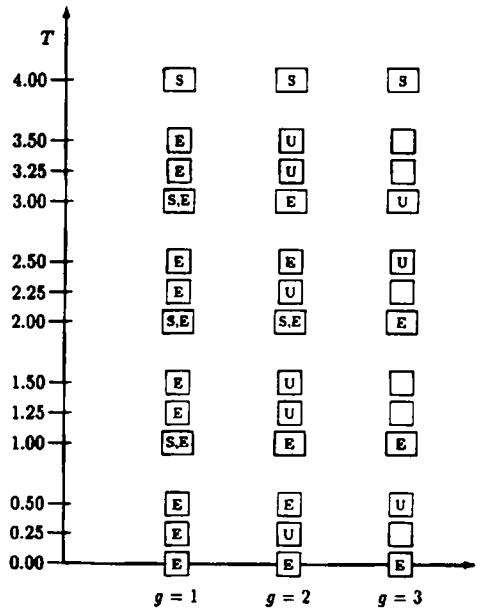

Figure 4. Evolution in time for multisequence time-stepping scheme. Example for three groups and three stages. U, update; E, evaluate residual; S, storing; $T = t/(CFL \cdot \Delta t_{b,\min})$

thousand time steps of the basic scheme the solutions of the two methods were identical at the same physical time, which confirms the time consistency of this concept. The measured gain in computational time was a factor of six compared with the basic scheme. This means that the multisequence Runge–Kutta scheme is much more effective for unsteady problems than the scheme with a variable number of stages for the same example (factor of two).

A higher number of groups would increase the speed-up factor of the multisequence Runge–Kutta scheme, since most of the nodes still reside in the last group. On the other hand, the time accuracy would then become critical. It should be mentioned that the speed-up factor is strongly dependent on the test case.

Comparisons with other time integration schemes. In comparing the efficiency of the proposed method with what is attainable by other enhanced time integration schemes, it is essential to note that the present multisequence Runge–Kutta scheme reduces the local stiffness and a gain is achieved only if the allowable time steps and cell sizes, vary essentially.

The scheme presented here can be compared with a similar approach, namely domain splitting for explicit schemes as proposed by Löhner *et al.*[13] The basic idea of this scheme is to split the whole domain into subdomains where a common time step is used. To achieve time consistency, the neighbouring domains are coupled via overlapping zones.

In contrast with domain splitting, the present multisequence Runge–Kutta scheme acts on each node locally and contains implicitly the time coupling between neighbouring nodes owing to the synchronization concept. The advantage is that the grouping of nodes and edges no longer requires any overlapping zones or special treatment on domain boundaries. Therefore the implementation is relatively simple and very flexible in terms of local time step restrictions. The computational overhead is so low that the grouping of nodes and edges can be done prior to each global time step, which enables the use of this algorithm for dynamic meshes.

Multigrid schemes are able to increase the overall efficiency, but the number of grid levels, and consequently the convergence, is restricted in unsteady computations for reasons of accuracy in time. Besides, additional convergence losses of multigrid schemes arise for strongly anisotropic grids, a typical effect in computations of high-Reynolds-number flows. Numerical experiments with a time-accurate FAS multigrid method for a similar problem (vortex street behind a cylinder) but on structured grids have shown a speed-up factor for multigrid of about two against a single-grid, basic five-stage Runge–Kutta scheme.[14]

Implicit time integration schemes also reduce the problem of local stiffness and allow a much larger time steps than explicit schemes. On the other hand, the time accuracy can be impaired in regions with small physical scale lengths where the local *CFL* number is large. A decrease in the global time step in such cases would reduce the efficiency of the implicit method.

4. MESH GENERATION AND ADAPTATION

Unstructured mesh generation is a mixed discrete–analogue optimization problem. The number of points and their connections to triangles are discrete; the position of the points can be considered as the analogue part. Such optimization problems are approximated by interchanging solutions of the analogue and discrete parts.

The generation consists of several tools, as described in the following, which act together on a closed triangulation. The process might be interrupted for a flow computation and continued for a following adaptation. Thus the steps of the mesh optimization problem are common for the generation as well for an adaptation of the mesh.

The applied method is a field method. The very first step is the generation of a mesh consisting of a triangulation between given boundary points of the domain. This input is produced by a rising bubble triangulation algorithm.[15]

4.1. Generation tools

Local mesh density. This is represented by the locally preferred length G_k of the edges of a triangle and is stored per node.

For boundary points this quantity is the length of the longer of the two boundary edges around such a point multiplied by a statistical factor.

For interior points a smooth variation in the size of triangles within the boundaries of the domain is desired. This requires the solution of a boundary value problem for the local mesh density. Since the boundary points and edges of the domain are known at the beginning of the generation process, the function G_k can be computed in advance; it does not therefore take part in the optimization process. However, there is no mesh yet to support the solution. Therefore the quantity G_k for the interior nodes is computed during the development of the mesh. This is done in a point Gauss–Seidel fashion by solving the equation

$$\nabla^2\left(\frac{1}{G_k}\right) = 0. \tag{11}$$

Another very common approach is to provide a mesh density function on a background mesh. This approach was not used, since more interaction would be required.

Point insertion. Additional points are inserted in the centre of all triangles under the following condition. At least one of the edges is longer than the local quantity G_k, defined above and none of the

neighbouring triangles has already been refined in the same insertion loop. The insertion is done in this sparse way in order to avoid a sudden point overflow.

Additional boundary points are inserted in the centre of the boundary segments. This may only be necessary in conjunction with stretching (see next subsection).

Edge reconnection. Two edge reconnection tools are employed, both based on diagonal swapping. The aim of the first one is to generate triangles which fit the Delaunay criterion. The second one reorganizes the mesh to obtain a triangulation with, as far as possible, six triangles around a common node. Since the two tools conflict, a compromise is taken.

Smoothing. Smoothing is done to increase the grid quality by recursively moving the points to optimal positions. The smoothing procedure is based on the circumcircle areas of the triangles and formulated as the minimization of a sensitive quantity T_S for all triangles of the mesh:

$$T_S = \sum_{i=1}^{ntr} \frac{A_c(i)}{(A_t(i))^w} (G_k(i))^{2(w-1)}, \tag{12}$$

where $A_c(i)$ is the circumcircle area of triangle i, $A_t(i)$ is the area of triangle i, ntr is the number of triangles, w is a weighting term for balanced triangle sizes and $G_k(i)$ is the average quantity G_k for triangle i.

Depending on the weighting exponent w, the triangle sizes or their forms are more important. For $w = 1$ the criterion takes care only of the forms of the triangles and disregards their sizes; thus zones of higher and lower density can appear. The problem arises only in the vicinity of points surrounded by a number of triangles, n_D, not equal to six. The reason for this problem is the fact that triangles of number unequal to six surrounding one point cannot be equilateral. To compensate this drawback, all the triangles consisting of points with different values n_D are geometrically transformed according to those numbers before being processed by the smoothing procedure. The transformation takes care of the fact that an optimal internal angle adjacent to a point is $\alpha_{opt} = 2\pi/n_D$. A triangle with three optimal internal angles would be transformed into an equilateral triangle; this form is considered optimal in the smoothing procedure.

The minimization is carried out by a pointwise, two-dimensional Newton method, iterating spatial derivatives of the quantity to be minimized to zero.

$$\frac{\partial(T_S(P))}{\partial X(P)} = 0, \qquad \frac{\partial(T_S(P))}{\partial Y(P)} = 0, \tag{13}$$

where $T_S(P)$ is the part of T_S influenced by the position of the node P. This is the sum of the sensitive amounts for all triangles having the node P as vertex. $X(P)$ and $Y(P)$ are the co-ordinates of the node P.

To ensure global minimization of T_S, the pointwise minimization is performed recursively in a point Gauss–Seidel procedure with unsorted address sequences.

This smoothing procedure consumes a considerable amount of computational time compared with other methods, e.g. Laplacian smoothers, averaging the location of neighbouring points. The advantages, however, are that there may not appear any invalid triangulations and that too flat triangles are avoided, which is important for unsteady flow computations with global time stepping.

4.2. Adaptation by virtual stretching

The aim of the generation procedure, as described above, is to generate a smooth homogeneous triangulation without any directionality. Only the density function was used to allow a smooth change between boundary lines with smaller or longer segments. Virtual stretching is used to generate flat

triangles and as an additional adaptation tool. The implementation is rather simple Mesh generation no longer proceeds in the physical plane but in a locally stretched one. Transformation back to physical co-ordinates yields meshes with anisotropic resolution or refined zones.

Stretching is internally represented by symmetric 2×2 matrices for each point of the developing mesh (generation) or the previous mesh (adaptation):

$$A = \begin{pmatrix} a11 & a12 \\ a12 & a22 \end{pmatrix}. \tag{14}$$

The stretching caused by such a transformation matrix is dependent on the angle Θ in the Cartesian plane. It is defined as the projection of the transformed unit vector in the direction of Θ onto this original direction:

$$S_A(\Theta) = \vec{e}(\Theta) \cdot (A \cdot \vec{e}(\Theta)). \tag{15}$$

Types of stretching

The way in which the stretching acts is best understood for the transformation of a unit circle.

Isotopic or scalar stretching shows no special orientation in the plane. A unit circle would be transformed into a larger circle. The stretching matrix is the unit matrix multiplied by a factor. The size of the triangles in physical space is reduced by this factor.

Anisotropic or tensorial stretching is the more general case. A unit circle would be transformed into any ellipse concentrically containing it. The scaling length of the triangles in physical space is reduced according to the length of the semiaxis of this ellipse.

Unidirectional or vectorial stretching is a special case of anisotropic stretching. A unit circle would be transformed into an ellipse tangent to the unit circle in the direction of the smaller semiaxis.

Computing the stretching matrices

Suppose that the mesh resolution in physical space is to be locally increased by a given factor R_1 in the direction of Θ_1, while in the direction Θ_2 perpendicular to Θ_1 the resolution is to be increased by a factor R_2.

This requires local stretching of the domain by a factor $R_1 = S_A(\Theta_1)$ in the direction of Θ_1 and $R_2 = S_A(\Theta_2)$ in the direction of Θ_2 while generating the mesh.

Regarding the transformation of unit vectors in the two perpendicular directions, one obtains

$$R_1\vec{e}(\Theta_1) = A \cdot \vec{e}(\Theta_1), \qquad R_2\vec{e}(\Theta_2) = A \cdot \vec{e}(\Theta_2), \qquad \Theta_2 = \Theta_1 + \pi/2.$$

The matrix A that satisfies these equations reads

$$A = \begin{pmatrix} R_1 \sin^2(\theta_1) + R_2 \cos^2(\Theta_1) & (R_1 - R_2)\sin(\Theta_1)\cos(\Theta_1) \\ (R_1 - R_2)\sin(\Theta_1)\cos(\Theta_1) & R_1 \cos^2(\Theta_1) + R_2 \sin^2(\Theta_1) \end{pmatrix}.$$

Combining several stretching properties. Special care is taken to combine different local resolution requirements, since several features are considered. This is done by a two-dimensional maximization procedure. Input are the stretching matrices $A_1, A_2, \ldots$ from all features to be considered for the regarded location. The desired output of the pointwise maximization is a matrix A_m containing all stretching properties of the input matrices with minimized determinant.

$$S_{A_m}(\Theta) \geqslant \max(S_{A_1}(\Theta), S_{A_2}(\Theta), \ldots) \quad \text{for } 0 \leq \Theta < \pi, \quad \det(A_m) = \min. \tag{16}$$

The minimum determinant is desired to minimize the number of points in the later mesh, since the determinant of the stretching matrices is the relation of areas between the virtual and physical planes.

Because of the difficulties in solving this problem, a recursive routine is applied which is able to maximize two matrices in one step. If more than two stretching matrices have to be combined, the previous maximum is kept to be maximized with the next input matrix. Note that if more than two matrices are involved, the exact minimum possible determinant is not obtained.

Features causing stretching

Several features require different minimal resolutions in different directions. The list below is not complete, since only the features used for the present computations are presented.

Vicinity of walls. The boundary layer of viscous flows next to solid surfaces has to be resolved by flat triangles oriented along the surface. They are generated using anisotropic stretching in the region next to the boundaries. Each boundary edge of a solid surface contributes with unidirectional stretching in its normal direction. Nodes are influenced according to their position relative to the edges and all contributions affecting a node are maximized with the maximization procedure.

The 'vicinity of walls' criterion is already used at the first generation of a mesh. To guarantee a proper resolution of the boundary layer during later adaptation, the stretching in the vicinity of the wall is remained.

Error estimation for shear flows. The discretization error for viscous terms is dependent on the spatial change in these terms. This means that not the shear situation itself has to be resolved carefully, but its first derivative in space. Note that this is not true for unsteady spatial changes, e.g. shear layers when computing inviscid flows. Vorticity is a good indicator of a shear flow situation. Its gradient vector is therefore evaluated per node and used as unidirectional stretching.

High-resolution spots. These may be placed interactively to increase the mesh resolution locally. The spots are introduced either by additional isotropic stretching, i.e. by simply multiplying the local stretching matrices by a factor, or by additional consideration of isotropic domain stretching within the maximization procedure. In the first case the refinement caused will approximately retain the aspect ratio of the triangles. In the second case the mesh resolution is only influenced in the directions in which the stretching of the spot is larger than the stretching caused by other features. Spots are free in their geometry; they may be circular around points, be a thickened line or cover a polygon and they may be smoothed out on their boundaries.

5. ADAPTATION FOR UNSTEADY FLOWS

5.1. Principles of unsteady adaptation

Since the generation and adaptation for a steady mesh comprise a process that can be interrupted and continued at any time, a simple version for an unsteady adaptation would be a continued meshing process while the flow computation advances. Some slight variations in the smoothing procedure even made it able to shift the points to optimized positions for longer distances; therefore a procedure able to take out points would not be required. Although this method produced very nice meshes, the computational results were poor. The reason is that all points are shifted always to new positions. Since the flow solver has not (yet) been formulated to discretize the equations on moving grids, an interpolation from one mesh to the other has to be performed. These repeated interpolations, presently

linear on triangles, act as a strong second-order damping. Thus dissipation is higher than without adaptation.

One way to solve the problem is to use a higher-order interpolation. Another way is to declare a set of points to be saved, i.e. they are never taken out or shifted. In our case the points of the non-adapted mesh were chosen. Adaptations are made using additional points. A shifting of these points throughout the mesh is impossible owing to the rigidity of the saved points. Therefore a procedure to take out additional points is required. Reconnections via diagonal swapping are still allowed even if saved points are involved.

Taking out points. In contrast with point insertion described at the beginning of the subsection, a take-out procedure is formulated. If a point is surrounded by edges that in stretched space are shorter than the local quantity G_k multiplied by a statistical factor less than unity, an attempt is made to take out this point. All edges previously connected to that point will later be connected to its nearest neighbour. In rare cases the so-produced connectivity may contain triangles with negative orientation. If this error cannot be rearranged immediately, the original connectivity is restored and the point remains.

The combined procedure. Mesh adaptation and flow computation run in a combined algorithm. Because data structures differ, intermediate routines are required as translators. The computations start on an initial mesh whose points will be saved. A previous solution on the non-adapted mesh is useful but not essential. The flow solver and the remeshing part work interchangeably, i.e. a mesh adaptation is performed in intervals. These intervals are chosen in such a way that the solution may not run out of adaptivity.

Around 100 adaptations are performed per flow cycle. Such an adaptation is a more or less empirical action sequence of the tools described. Typically it is useful to first perform some loops taking out points, inserting new points, reconnecting edges and smoothing for a few steps. After that some further loops without taking points out or inserting new ones but with some more smoothing steps are performed.

The whole recurrent process is outlined as follows.

(a) Apply flow solver for a set of time steps.
(b) Compute stretching according to actual solution.
(c) Store old mesh and solution.
(d) Make a new mesh (adapting the old one).
(e) Sort new mesh to improve memory access.
(f) Interpolate variables for additional points (presently linear on triangles of old mesh).
(g) Translate data structures for flow solver.

Items (b)–(g) make up the adaptive part of the computations.

5.2. *Computational examples*

The test example describes the subsonic flow at $Ma_\infty = 0{\cdot}3$ around an aerofoil with an angle of attack of 0°. The flap is extended to 10°. No turbulence model has been implemented yet; the Reynolds number is chosen as $Re_\infty = 10^4$. This laminar test case therefore has no technical importance but is a well-suited example to demonstrate the ability to treat complex geometries and unsteady flow features with the present algorithm.

The flow next to the aerofoil is nearly steady; in particular, the upper and lower separation points do not vary significantly. In the gap between foil and flap a flow is generated, directed to the upper side.

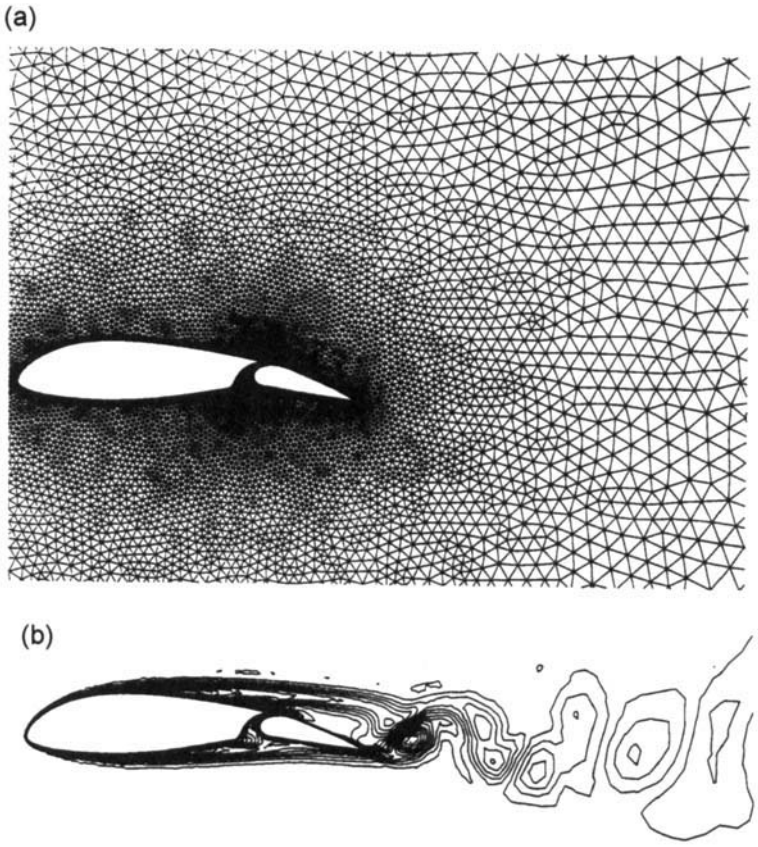

Figure 5. Laminar, separated flow around aerofoil with flap ($Ma_\infty = 0{\cdot}3$, $Re_\infty = 10^4$). Computation on static mesh. (a) Static mesh with around 16,000 grid points (not all shown here). (b) lines of constant vorticity at fixed time, computed on static mesh

This flow interacts like a jet with the separated shear layer of the upper side. At the trailing edge of the flap, vortices start rolling up and generate a periodical vortex street farther downstream.

Computations on static grids. The usual way to compute unsteady, vortical flows is to use time-independent, static grids. Then the mesh has to be sufficiently fine for resolving the moving flow at each location at each time. The present static mesh is preadapted to fit stationary features such as the boundary layer regions around the body. Figure 5(a) shows the non-adapted mesh with around 16,000 grid points in total. The corresponding computed lines of constant vorticity around the aerofoil are presented in Figure 5(b). The solution presents all essential features of the flow, but finer details in the shear layers and in the vortex core are smeared.

Computations on dynamic grids. Despite the relatively high resolution in Figure 5, some details of the flow, in particular the moving parts, are not sufficiently resolved. It would therefore be desirable to adapt the details using a moving, dynamic mesh such that zones of fine resolution move with the critical flow structures.

A combination of a static mesh and a dynamic mesh with time-dependent grid cell distributions is used here. The static mesh is the same as used before and shown in Figure 5(a). The dynamic grid

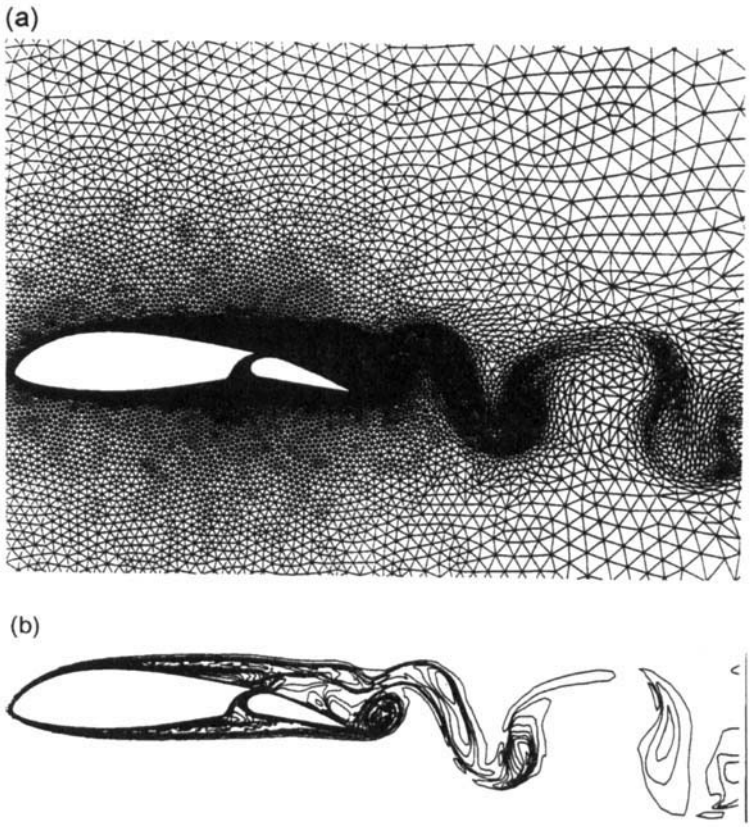

Figure 6. Computations on combined static and adapted, dynamic mesh. Legend as in Figure 5. (a) Adapted mesh (static and dynamic mesh) at fixed time with around 16,000 static and 6000 dynamic grid points (not all shown here). (b) Lines of constant vorticity at fixed time, computed on time-dependent, adapted mesh of (a)

consists of around 6000 additional grid points. Around 100 adaptations are performed per flow cycle. Figure 6(a) shows the adapted mesh at a fixed time within one flow cycle. The corresponding lines of constant vorticity are plotted in Figure 6(b). Comparison with Figure 5(b) indicates clearly the improvement with unsteady adaptation, where the details of the vortex flow are much better represented. Figure 7 demonstrates the time dependence of the adaptive grid. These figures show a detailed portion of the mesh next to the flap at four different times within the flow cycle. These are four of around 100 meshes required for one flow cycle. It is interesting to see that the unsteady flow features can be qualitatively represented by the grid cell distributions of the adapted mesh.

The CPU time of the adaptation procedure is only a few per cent compared with the basic Runge–Kutta scheme. In combination with the much faster multisequence Runge–Kutta solver the adaptation procedure takes nearly 30% of the total CPU time.

6. CONCLUSIONS

An adaptive method for unsteady solutions of the compressible Navier–Stokes equations is presented. A finite volume discretization is applied for unstructured, triangulated grids. The mesh generation and

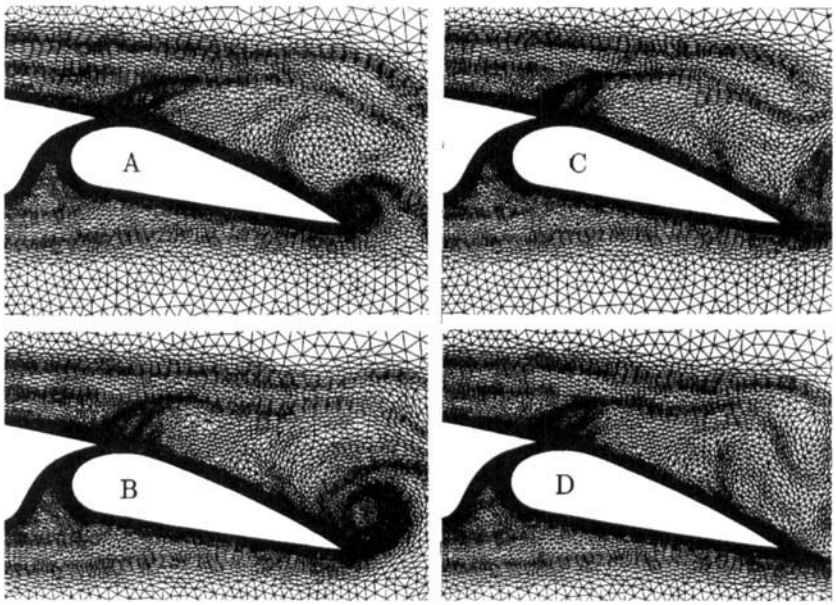

Figure 7. Sequence of adapted meshes at four different times within one flow cycle. Legend as in Figure 5

adaptation are based on a common algorithm, where the mesh adaptation is a continuation of the generation process. The adaptation concept, called virtual stretching, allows simultaneous adaptation with respect to different criteria.

The method enables the computation of complex, unsteady flow features with an efficient explicit Runge–Kutta multisequence scheme, as presented in this paper. High resolution was achieved by using dynamic grids moving with the unsteady flow features. Turbulence modelling and extension to three-dimensional flows are goals of future developments.

REFERENCES

1. R. Löhner and J. D. Baum, 'Numerical simulation of shock interaction using a new adaptive h-refinement scheme on unstructured grids,' *AIAA Paper 90-0700*, 1990.
2. T. J. Baker and A. Jameson, 'Improvements to the aircraft Euler Method,' *AIAA Paper 87-0452*, 1987.
3. J. Peraire, L. Formaggio, K. Morgan and O. C. Zienkiewicz, 'Finite element Euler computations in three dimensions,' *AIAA Paper 88-0032*, 1988.
4. D. J. Mavriplis and A. Jameson, 'Multigrid solution of the Navier–Stokes equation on triangular meshes,' *AIAA J.* **28**, 1415–1425 (1990).
5. P. K. Prabhu, J. R. Stewart and R. R. Thareja, 'A Navier–Stokes solver for high speed equilibrium flows and application to blunt bodies,' *AIAA Paper 89-0668*, 1989.
6. K. Morgan, J. Peraire and J. Peiro, 'Unstructured grid methods for compressible flows,' in *Special Course on Unstructured Grid Methods for Advection Dominated Flows, AGARD Rep. R-787*, pp. 5-1–5-39, 1992.
7. R. Löhner, K. Morgan, J. Peraire and M. Vahdati, 'Finite element flux-corrected transport for the Euler and Navier–Stokes equations,' *Int. j. numer. methods fluids*, **7**, 1093–1109 (1987).
8. M. Mallet, 'Adapted finite element methods for hypersonic reentry problems,' in *Third Joint Europe/U.S. Short Course in Hypersonics*, RWTH, Aachen, 1990.
9. R. Vilsmeier and D. Hänel, Adaptive solutions for compressible flows on unstructured, strongly anisotropic grids,' in C. Hirsch, J. Periaux and W. Kordulla (eds), *Computational Fluid Dynamics '92*, Vol. II, Elsevier, Amsterdam, 1992, pp. 945–952.
10. M. Gehle, D. Hänel and R. Vilsmeier, 'Adaptive grid methods for viscous flow,' *Proc. 5th Int. Symp. on Computational Fluid Dynamics*, H. Daiguzi, Japan Society of Computational Fluid Dynamics, pp. 247–252, Sendai, August–September 1993.

11. D. J. Mavriplis, 'Solution of the two-dimensional Euler equations on unstructured triangular meshes,' *Thesis*, Princeton University, 1987.
12. M. Pervaiz and J. Baron, 'Spatiotemporal adaption algorithm for two-dimensional reacting flows,' *AIAA J.*, **27**, 237–252 (1989).
13. R. Löhner, K. Morgan and O. C. Zienkiewicz, 'The use of domain splitting with an explicit hyperbolic solver,' *Comput. Methods Appl. Mech. Eng.*, **45**, 313–329 (1984).
14. M. Meinke and D. Hänel, 'Time accurate multigrid solutions of the Navier–Stokes equations,' *Int. Ser. Numer. Math.*, **98**, 289–300 (1991).
15. R. Vilsmeier and D. Hänel, 'Generation and adaptation of 2-D unstructured meshes,' in A. S. Arcila, J. Hauser, P. R. Eiseman and J. F. Thompson (eds.), *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, North-Holland, Amsterdam, 1991, pp. 55–66.