ALGORITHMS FOR THE AUTOMATIC GENERATION OF 2-D STRUCTURED MULTI-BLOCK GRIDS

Thilo Schönfeld
CERFACS
42, avenue Gustave Coriolis
F-31057 Toulouse Cedex
France

Per Weinerfelt
Department of Mathematics
Linköping University
S-581 83 Linköping
Sweden

Carl B. Jenssen
Dep. of Industrial Mathematics
SINTEF - SIMa
N-7034 Trondheim
Norway

SUMMARY

Two different approaches to the fully automatic generation of structured multi-block grids in two dimensions are presented. The work aims to simplify the user interactivity necessary for the definition of a multiple block grid topology. The first approach is based on an *advancing front method* commonly used for the generation of unstructured grids. The original algorithm has been modified toward the generation of large quadrilateral elements. The second method is based on the *divide-and-conquer* paradigm with the global domain recursively partitioned into sub-domains. For either method each of the resulting blocks is then meshed using transfinite interpolation and elliptic smoothing. The applicability of these methods to practical problems is demonstrated for typical geometries of fluid dynamics.

INTRODUCTION

Structured multi-block grids are distinguished by regular quadrilateral cells (hexahedras in 3-D) which allow to obtain solutions of good accuracy. However, this high degree of regularity is too stiff when considering geometrical complex shapes. The direct mapping of the physical domain onto one *single* computational domain becomes increasingly complicated. The discretized domain must be partitioned into a set of sub-domains. For each of the resulting so-called blocks a grid is generated separately. The first step in the generation of such a multi-block grid is the definition of the block topology. An overall structure for the arrangement of the blocks, together with internal connectivity information and local coordinate orientations must be set up. Traditionally the block topology is set up manually, a pre-processing task that can quickly become difficult with rising complexity. This led us to the idea of replacing the user interactivity by an automatic generation of the blocks. Examples for references on techniques for the automatic meshing of quadrilateral grids are those of [2] or [8].

The aim of this work is to develop methods for generating structured multi-block meshes automatically in order to reduce the amount of pre-processing required for multi-block calculations. Multi-block solvers can then be applied to complex geometries with almost the same level of flexibility as solvers based on unstructured methods. Our original interest is to provide meshes for problems in fluid dynamics, but automatic mesh generators are highly demanded in many other fields.

The overall idea behind our first block-generation method is to combine the advantages of a structured multi-block grid with a technique commonly used to generate unstructured grids. In an unstructured environment the rigidity of the structured (i, j)-index system is broken and replaced by a system of pointers. Since the generation of an unstructured grid is done locally, and since only a 'single block' is required for a complex configuration, no special topology needs to be defined. We propose a block generation algorithm

C-7

4

that is based on the *advancing front technique* (AFT), which has been applied e.g. by Peraire et al. [6] or Löhner et al. [5] to generate small triangular elements (that directly form the final grid cells). Here, we use the advancing front technique to generate large elements that serve as blocks and thus have to be covered by mesh lines in order to obtain the final computational grid. The basic ideas behind the AFT algorithm are retained, with the exception of two principal modifications: Firstly, we create *quadrilaterals* instead of the originally generated triangles and, secondly, we generate *blocks* of maximal possible size rather than small *grid cells.* Compared to the basic AFT for triangles, the present algorithm for the generation of rectangular elements is more complicated, partly because of attempts made to ensure robustness and the prevention of distorted elements. The additional computational costs are easily justifiable, since much less elements are required compared to a complete unstructured mesh.

Earlier results with an application of AFT as a multi-block generator are presented in [1], where so-called 'micro-blocks' are created by forking triangles in an unstructured triangular mesh. However, this technique results in skewed blocks which degrades the accuracy of a flow calculation on the resulting meshes. It therefore seems advantageous to replace the triangular elements by the direct generation of quadrilateral blocks. This modified AFT-approach for the generation of quadrilaterals instead of triangles was first presented in [7] and is in the following referred to as 'method 1'. Although fully automatic, both of these AFT-approaches are based on underlying unstructured techniques. The drawback with these methods is that the topology is difficult to control which might lead to undesirable effects on the grid.

These shortcomings led us to the second approach ('method 2') [3], which, instead of building blocks as with AFT, is based on a successive partitioning of the domain of interest. This permits a better control of the topology. With this approach, the initial computational domain is successively sub-divided, until each sub-domain (block) is considered acceptable, according to certain criteria, for the generation of a structured mesh. The desired properties of such a block are for example that it should consist out of four corners where each corner should be as close as possible to a right angle. The partition procedure consists of first flagging boundary nodes as *cut required*, *cut permitted* or *cut not permitted* (corner node) and then chosing an optimal cut between two nodes. The success of this algorithm depends on the definition of the function measuring the quality of a cut. Though there are obviously several choices for this function, they all have to take into account various criteria such as the number of corners in the created sub-domains or the total number of boundary nodes still requiring a cut.

METHOD 1: THE ADVANCING FRONT METHOD

The advancing front technique has obtained its name from a 'front' that 'advances' (travels) through the domain of discretization. This front, often referred to as the actual or current front, is defined as a closed curve of assembled line segments which changes continuously during the generation process. After each cycle of the algorithm one (or several) elements are generated, often by the simultaneous creation of a new grid point. A description of the underlying philosophy of our modified AFT-algorithm is given hereinafter.

AFT for Rectangles

Background grid: The so-called background grid, used for interpolation purposes only, is created interactively and consists of large triangles which cover the complete domain of discretization (Fig. 1). At the vertices of each triangle, stretching parameters are stored. These values are user-defined before the generation process starts and enable the block size to vary throughout the domain. If desired, blocks of equal size can be obtained by using the same value at all the vertices. The information required at any point in the domain is then obtained by linear interpolation of the stretching values at the vertices. In order to facilitate the creation of the background grid and to minimize the searching and interpolation procedures, the number of elements is kept as small as possible.

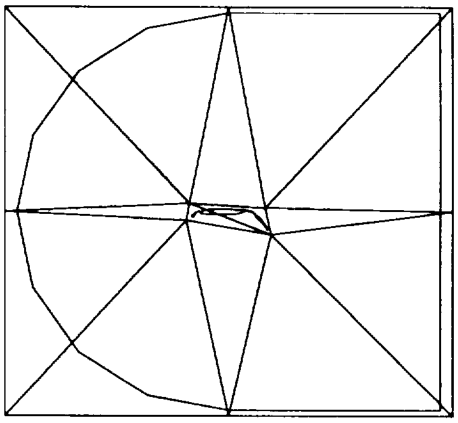

Figure 1: **Background grid for multi-element airfoil.**

Surface discretization: Unlike for the triangular AFT [6], in the present method the background grid is not employed for the discretization. Since large blocks are generated rather than the final mesh, a relative coarse approximation of the solid surfaces is sufficient.

In a standard application of the block generator, solid surfaces are discretized by equally-spaced points. These nodes belong to the final grid, and few of them are actually used in order to obtain a coarse discretization. In an alternative approach, all of these points are used to describe the geometry, thus leading to a much improved approximation of the solid surfaces. The supposed disadvantage of the latter technique for the purpose of the block generation is neutralized by the method of merging neighboring faces.

Initial front: By means of the background grid the starting front is set up. This consists of contiguous nodes on given discretized curves connected by straight lines. These line segments (in the following notation called 'faces') are referenced by integer arrays, with two indices describing the start and end-point of the face. In order to enable the use of one single closed curve (once the parts of the front that are initially separated are connected), the line segments on interior surfaces are stored in clockwise orientation (defined by the order in which the data is entered), while the faces on exterior boundaries are oriented in a counter-clockwise direction (or vice versa). The strict preservation of this rule throughout the entire generation process is an essential part of the algorithm.

The Block Generation Algorithm

Compared to the generation algorithm for triangles, a large number of additional requirements are needed to control the automatic generation procedure. The robustness of the algorithm is one of the major considerations. The algorithm consists of the following principal steps, which are partially identical with those of the traditional triangle algorithm:

(i) As in the triangle algorithm, a choice is made for an initial base for the next element to be generated. The 'smallest length' criterion is used.

(ii) Next, the important means to merge faces comes into effect. The angle β between the chosen initial basis and the faces next to it (on both ends) are checked. The initial base-line, together with the faces that are to be merged, form the final basis with end-points A and B (Fig. 2). The pointers that indicate the positions of the adjacent nodes are then examined. In case $\overline{AB}$ belongs to a closed polygon with maximal six corners that cannot be sub-divided into two quadrilaterals, the items (iii) to (vi) listed below are skipped and the new element is set up directly.

(iii) The third major step is to find the position of the *two* ideal points C_{0A} and C_{0B}. By 'ideal' we denote that point which is chosen if none of the nodes of the current front is selected. One ideal point for each the start and end-point of the basis has to be found. Since this task is more difficult compared to the standard AFT algorithm (with only one ideal node per element), below we give a more detailed description.

(iv) When the ideal points C_{0A} and C_{0B} have been selected, we determine all the potential nodes C_i that belong to the actual front and which lie inside a circle of given radius r, with center C_{0A} (reasonable value $r = 1.7 \times \delta_A$). For reasons of simplicity, we restrict our attention to the point A; the procedure for B is analogous. The coordinates of the flagged points are ordered according to their distance from point C_{0A}; four 'reserve' nodes C_R are added to the end of the list.

(v) In this step, the connecting point C_A (C_B for point B) is determined. This point must satisfy all of the requirements described below.

(vi) Once a valid point has been designated for both nodes A and B, a new quadrilateral element is defined by the vertices A, B, C_B and C_A. Four principal configurations are possible and the correct one has to be determined.

(vii) After the definition of a new element, the corresponding pointers are updated and the number of remaining faces is checked. If any faces remain, the whole procedure is repeated by starting at item (i); if none remain, the block generation is terminated.

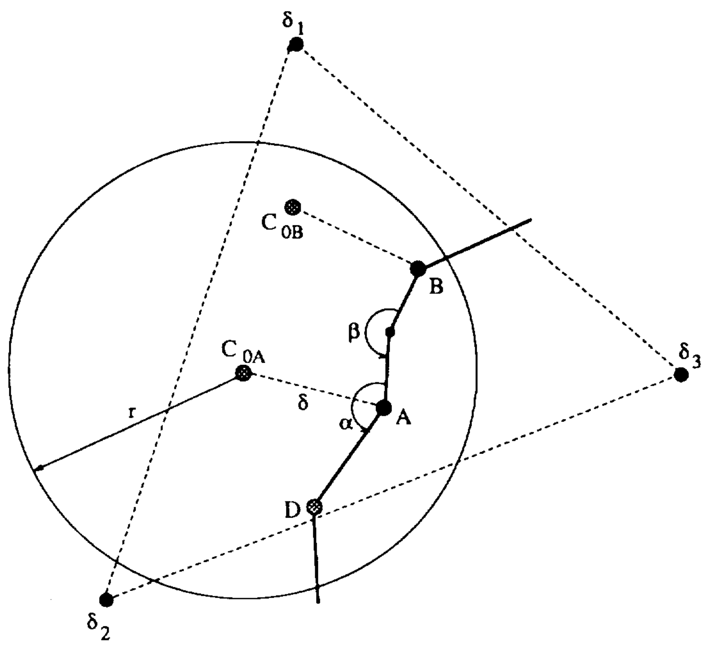

Figure 2: General notation for rectangle algorithm.

Face merging: This tool to reduce the number of blocks is one of the essential ingredients incorporated into our advancing front algorithm. Two neighboring faces may be merged to form one long face whenever the angle β between them is 'smooth', i.e. close to 180° (Fig. 3). The use of angles (which are non-dimensional) allows to circumvent the (heuristic) definition of the delicate threshold parameter that is necessary when working with the curvature (which depends on the geometric scaling of the configuration).

Typical values that define a smooth angle lie between 160° and 200°, although these threshold values may be chosen close to 180° at solid walls. Note that not only the base side may consist of several merged faces, but also the remaining three sides of each element. Indeed, the combination of faces opposite to the basis has been found to be efficient.

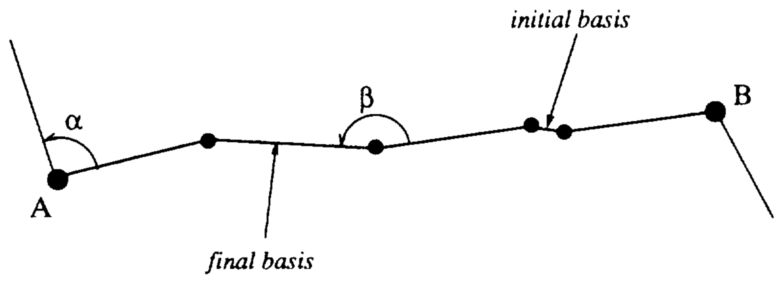

Figure 3: Face merging.

Finding the ideal points: For each end-point A and B of the final basis, two ideal points must be selected. Different principal strategies are used depending on whether the angle α between the base side $\overline{AB}$ and its neighbors is smaller or larger than 130°. This heuristic value, as well as all the other characteristic angles, has proved to be the most efficient when developing the program (efficient in terms of both algorithm robustness and general validity for any given configuration). We distinguish between the following cases (Fig. 4):

(i) $\alpha > 130°$: The angle is large enough to be sub-divided without creating skewed elements. One new point is set up. The stretching length δ of the vector $\overrightarrow{AC_{0A}}$ is obtained from the background grid. The vector direction again depends on α:

  (a) $130° < \alpha \leq 250°$: The angle is divided into two equal halves, i.e. the direction of $\overrightarrow{AC_{0A}}$ is the mean value of the normal directions to $\overline{AB}$ and $\overline{AD}$, where D denotes the other node adjacent to A on the present front.

  (b) $\alpha > 250°$: A sub-division into two halves would lead to skewed blocks. The adjacent face is then disregarded and $\overrightarrow{AC_{0A}}$ is chosen to be orthogonal to $\overline{AB}$.

(ii) $\alpha \leq 130°$: Again, the subdivision into two halves would lead to an element with a distorted shape. The nearest neighbor of A is chosen as an ideal point; the length of $\overline{AD}$ is a given length δ.

Since we generate quadrilateral elements, no limitation on the stretching length δ is necessary (contrary to the classical AFT). Long stretched blocks are allowed.

Criteria for point validation: In this section we give a survey of the criteria to be satisfied by a point C_i in order to be a valid node C_A.

(i) The point under consideration C_A does not belong to any of the merged faces that form the base edge.

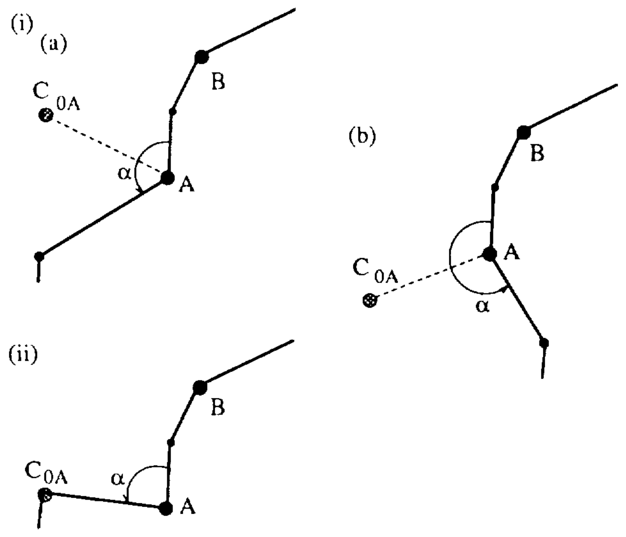

Figure 4: Checking of angles.

(ii) In the case when the point C_A is one of the neighboring nodes, the angle α must not be too large. This check is performed in a successive way for all the contiguous points of A.

(iii) The angle between $\overline{AB}$ and $\overline{AC}_A$ obeys $50° \leq \angle BAC_A \leq 130°$.

(iv) The triangle $\triangle BAC_A$ has a positive area, i.e point C_A lies in the interior of the domain. Here the strict maintenance of the orientation plays an important role. This condition corresponds to the first requirement of the standard AFT algorithm.

(v) The second well-known criterion of the classical AFT approach checks for any intersections of the line $\overline{AC}_A$ with existing faces. Note that crossings with *all* existing faces must be examined, not just those with edges on the *current* front.

These tests are carried out for the flagged points of both end-nodes A and B of the basis. The list of reserve points guarantees that at least one node fulfills all of these five conditions. Since the check for node B follows that for A, the designated node C_A is already known. Additional validity requirements must be imposed on point C_B:

(vi) Trivial: Node C_B must not be identical with C_A.

(vii) There must be no intersections between the potential line $\overline{C_AC_B}$ and any of the existing edges.

Creation of a new element: During one 'quadrangulation' step exactly *one* new element is generated. This characteristic feature of the algorithm maintains a clear structure for the generation process. The type of this element is determined by the kind of the new nodes C_A and C_B. In the following, we describe the principal configurations that can appear (see Fig. 5):

(i) In the ideal case, one new node is placed in the domain for either node A and B. Three new faces are formed. Points C_A and C_B are identical with C_{0A} and C_{0B}, respectively (or one of the reserve nodes).

(ii) If a neighbor of A (or B) is found, a new point is chosen for B (or A). One new node and two faces are created. The faces that form the side $\overline{AC_A}$ are deleted from the actual front. In the special case when an adjacent node has been chosen for both A and B, no new point is set up and the connection line $\overline{C_AC_B}$ forms one new side.

(iii) If (as in the previous case) one neighbor of A has been chosen, but the point C_B is located on another part of the front, no new node is set up, and one new side is created. All the faces that form $\overline{AC_A}$ are removed from the current front, as well as the faces between C_A and C_B. Similarly for point B. In the special case when both C_A and C_B are located on another part of the front, but neither is a neighbor of either A or B, no new node but two new sides are defined.

(iv) In the final case, an element with either quadrilateral, pentagonal, or hexagonal shape is chosen. The reason for the implementation of this special case was to reduce the computational costs by avoiding the node validation verifications. Starting at the base side, all the faces that form the polygon are discarded.

For all of the cases mentioned above, the faces which form the base side are removed from the list of actual front faces. The new faces are added to the current front, hereby conserving the strict orientation rule.

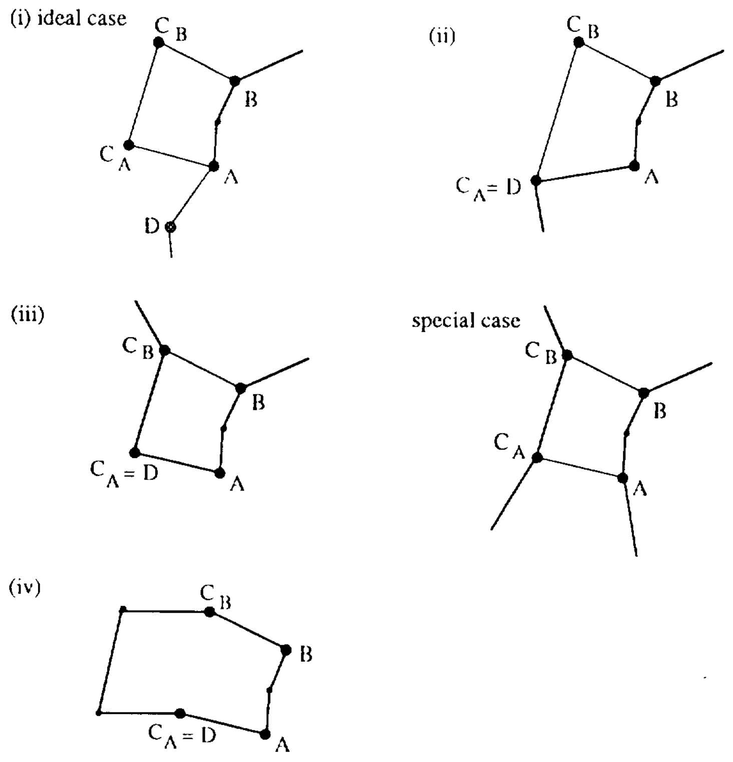

Figure 5: Potential new elements.

The Pointer System

The data structure of the present block generator employs a pointer system. Two pointers are necessary to keep track of the block generation process. The first relates the nodes to the element faces. For each face, this array contains: in its first position, the index of the start point; in the second position, the end-point; in a third position, the face boundary condition type is stored. A second pointer contains the inverse information and links the faces to the nodes (though this is no longer of importance when the corresponding face is deleted from the actual front).

Additional pointers are required to provide the necessary information for both the mesh generator and the flow solver (note that these pointers are independent of the block generation procedure and thus are not essential for the algorithm). An element-to-face pointer relates the faces to the four block sides. In a strict rule faces 1,2 and 3,4 are pairs of opposite faces of each element. The inverse pointer links the elements to the faces. The third output pointer is the face-to-node pointer. Finally, the fourth pointer stores the number of merged points on solid wall faces (this information is necessary for the mesh generator).

Block Generator Post-Processing

The automatic generation procedure usually gives a larger number of blocks than desired, since the robustness of the algorithm is deemed to be more important than creating as few elements as possible. For this reason, a post-processing routine has been developed. After viewing the data given by the block generator, neighboring blocks may be merged manually. The second principal purpose of post-processing is to initialize the number of grid points per block-side.

METHOD 2: DIVIDE AND-CONQUER

The second of our methods has been developed with the experience gained by the previous approach and is based on a *divide-and-conquer* paradigm.

Block Generation by Successive Domain Partitioning

The basic idea behind this technique is to successively partition a given domain into sub-domains until all sub-domains are acceptable for mesh generation, according to certain criteria. This principle is illustrated by the recursive divide and conquer block creation procedure, **createblock**, written in a pseudo language:

```
procedure createblock(domain);
    if (domain not a block)
% partition the domain into subdomain 1 and subdomain 2
        partition(domain, subdomain 1, subdomain 2);
        createblock(subdomain 1);
        createblock(subdomain 2);
    end
end procedure
```

As can be seen from the code, we need a definition of a *block* and a *partition* procedure. The criteria defining a block should ideally be based on mathematical principles related to eg. the shape of the domain. Since it might be difficult to find a proper formulation, we have instead chosen heuristic criteria mimicking those used in an interactive block generation session. The desired properties of a block are hence:

(i) A block should have four corners.

(ii) Each corner should be as close to a right angle as possible.

(iii) Each side should have a limited curvature.

(iv) The area of the block should be as large as possible.

The main reason for considering these is that they support the generation of high quality meshes by simple mesh generation algorithms like transfinite interpolation and elliptic smoothing. We need, however, to modify items (i)-(iv) slightly in order to obtain robust criteria for the block partitioning. Here, robust means that the algorithm should terminate after a finite number of steps, i.e. any given domain should be split into a finite number of blocks. Criteria (i)-(iv) are also involved in the procedure of deciding if and how to subdivide a given domain, i.e. the main part of the partition procedure. Before formulating the final block criteria and the partition procedure, we start by introducing some notations and definitions.

The input data to the createblock routine is a 2-D domain with a boundary which is a closed polygonal curve with positive orientation as shown in Fig. 6.

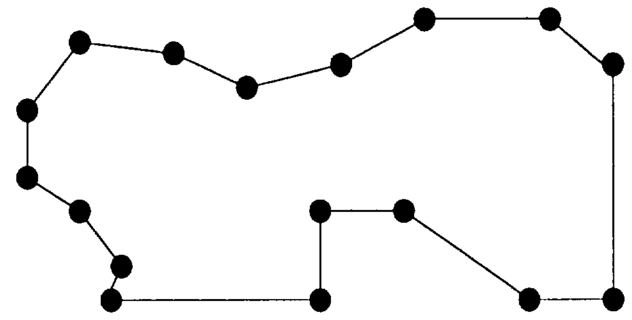

Figure 6: Sample domain.

First all vertices of the polygon, i.e. boundary nodes, are flagged. The flagging options are *cut required* (*cr*), *cut permitted* (*cp*) and *cut not permitted* (*cnp*). The first and foremost criteria when flagging the nodes is the size of the angle α between two neighboring line segments as shown in Fig. 7.

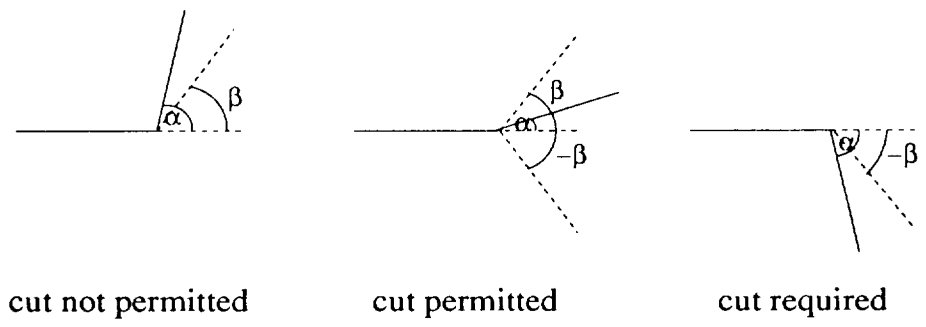

Figure 7: Flagging of nodes based on the angle.

Let β be a predefined fixed angle, $0 \leq \beta \leq \pi/2$, then a node is marked as *cr* if α fulfills $\alpha \leq -(\frac{\pi}{2} - \beta)$. If $-(\frac{\pi}{2} - \beta) < \alpha < \frac{\pi}{2} - \beta$ the node is marked as *cp*. Finally, when $\alpha \geq \frac{\pi}{2} - \beta$ the node is marked as *cnp*. A *cnp* node is also called a corner. Furthermore we need to introduce the notation *accumulated curvature*. The accumulated curvature between two corners is defined as the sum of the absolute value of the angles α. The reason for using this notation is that property (iii) above suggests a limitation of the curvature of a block side. Hence the accumulated curvature is computed for all nodes along the boundary and each time it exceeds a multiple of a predefined angle γ the actual node is marked as *cr* (see Fig. 8).

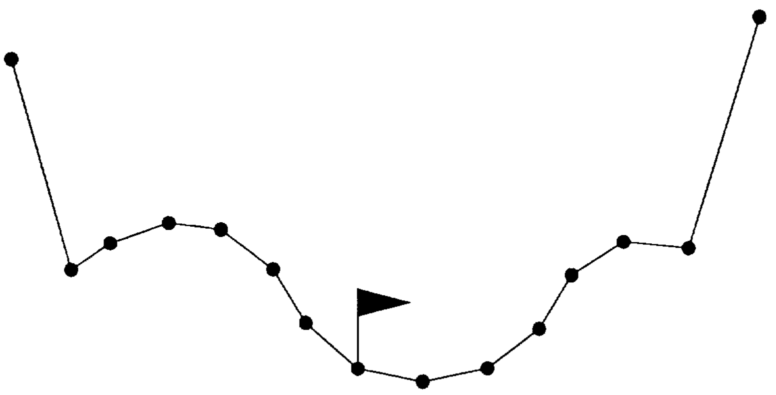

Figure 8: Flagging of nodes based on the accumulated curvature.

We can now proceed defining the modified criteria of a block. As mentioned above the final blocks should ideally have the desired properties (i)-(iv). Some of these are included in the partitioning algorithm, as will be seen later, and hence not be applied explicitly. The following criteria for a block are finally chosen:

(i) A block has 3, 4, or 5 corners.

(ii) A block does not contain any node flagged as *cr*.

The first requirement ensures that the blocking procedure terminates after a finite number of steps. This might not be the general case if we only allow blocks with four corners. As illustrated in Fig. 10 triangles and pentagons cannot be partitioned into quadrilaterals with a single cut, without splitting the corners, since splitting such blocks result in at least one new block with same number of corners. Thus obtaining a domain with three or five corners has to be accepted as a block in order to terminate the splitting of the domain. If a domain is not accepted as a block, it is passed to the partition procedure for further subdivision. We will now continue the description of the partition procedure. First, a set of points, between which a cut can be performed, has to be prescribed. The cut will here be restricted to the set of boundary nodes. Among all possible cuts only *allowed cuts* are considered. By an 'allowed' cut we mean a cut between a node marked as *cr* and a *cr* or *cp* node. The *best cut* is finally selected from the set of allowed cuts. Hence a domain, which is not a block, need to have at least one *cr* node in order to be split. If no such node exists the node corresponding to the smallest angle α is re-flagged to a *cr* node. Finally, additional boundary nodes, flagged as *cp*, are inserted to ensure an ample amount of possible cuts to chose from. These nodes are chosen such that no line segment is longer than a predefined characteristic length, l_{scale}, and so that there is at least one *cp* node between two *cnp* nodes.

We now proceed, by describing how to select the best cut among all legitimate cuts. The success of the algorithm depends crucially on the definition of best cut. Following the approach of Talbert and

Parkinsson [9], we define 'best cut' as that cut which minimizes a function measuring the quality of the two new domains created by the cut. This function is chosen as a linear combination of terms measuring the angles between the cut and the boundary, the distance from the cut to the boundary, the number of corners in the resulting subdomains, and whether or not both nodes defining the cut are flagged as *cr*. Thus our function f is defined as:

$$f = w_1\theta + w_2 s + w_3 n_c + w_4 r \tag{1}$$

where w_1, w_2, w_3, and w_4 are non-negative weights and

$$\theta = \min_{n=-1,0,1} |\alpha/\pi - n/2| \tag{2}$$

$$s = \max(0, (l_{scale} - l)/l_{scale}) \tag{3}$$

$$n_c = \begin{cases} 1 & \text{if number of corners} = 3 \text{ or } 5 \text{ in either of the two blocks} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$r = \begin{cases} 1 & \text{if only one node is flagged } cr \\ 0 & \text{if both nodes are flagged } cr \end{cases} \tag{5}$$

The angle measure (2) tends to encourage cuts that intersects the boundary either at an approximately right angle, or that makes a smooth transition from boundary to cut. For clearity, this is illustrated in Fig. 9.

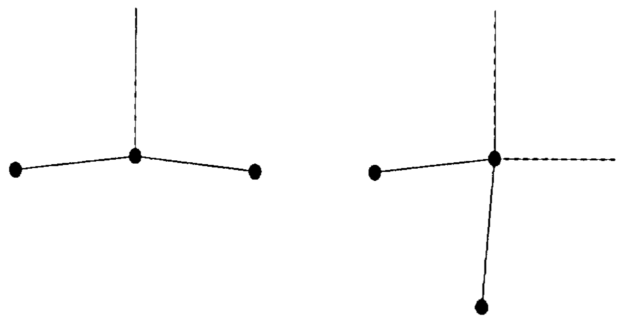
Figure 9: Preferred angles between boundary and cut.

The distance measure (3) is designed to avoid excessively thin blocks. Here the parameter l_{scale} is a typical length scale. The corner count (4) has the purpose of avoiding domains with three or five corners. The final measure (5) encourages cuts between two nodes that both are flagged as *cr*. This is intended to reduce the total number of cuts that has to be performed.

Finally, before the generated block topology can be passed to the mesh generator, it must be post-processed so that all blocks have four sides. This is achieved by a re-flagging of nodes wherever necessary. This re-flagging is again based on the angle between neighboring line segments as shown in Fig. 7. For

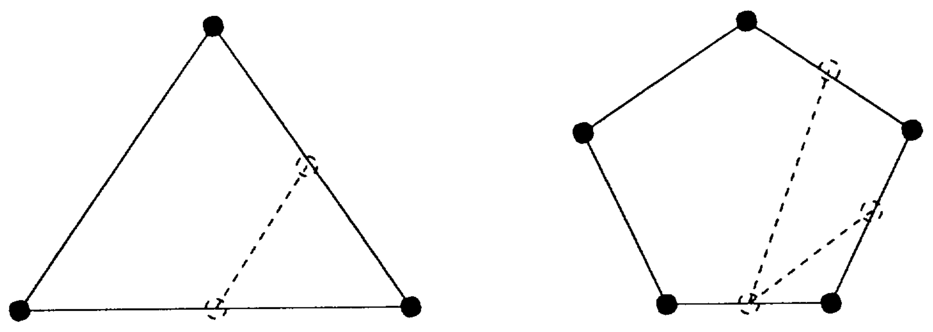
Figure 10: Subdivision of triangles and pentagons.

pentagons the corner node with the smallest angle is ignored, while for triangles, the node with the largest angle not already defined as a corner, is flagged as a corner. Note, that only the definition of corners or sides changes, while the actual shape of the block remains unchanged.

The output from the block generator then consists of blocks as shown in Fig. 11. Each block has four sides, where each can be sub-divided into a number of edges. The different edges correspond to different boundary conditions. All boundary conditions, except internal boundaries (i.e. boundaries which are shared by two blocks), are supplied by the input geometry. For internal boundaries the condition is defined simultaneously with the creation of new internal edges.

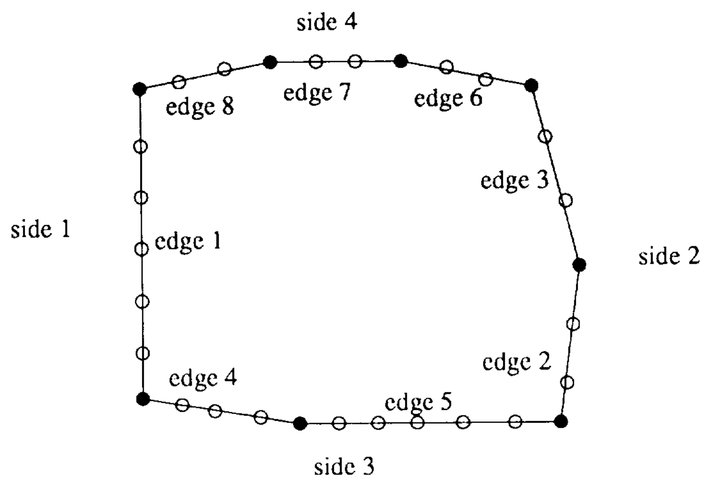

Figure 11: Information created by the block generator.

MULTI-BLOCK MESH GENERATION

The second part in the multi-block generation method consists of generating a structured mesh in each block. This part is common to both of the methods described above and the same mesh generation tool is used. Before a mesh can be generated, additional information has to be created by a block-grid generator interface.

The Block–Grid Generator Interface

The block generator provides topology information in the form of node-coordinate, edge-node, and edge-side-block registers together with the type of boundary condition on each edge. From these pointers all necessary information for the grid generator can be obtained. Part of the data can be used directly while others are retrieved from an interface program.

Before generating the mesh the number and the type of boundary conditions, the connectivity to other blocks and the number of grid points on each of the four block sides must be determined. The crucial task is to compute the number of grid points on the edges from which we get the number on the block sides. If the blocks are meshed completely independent of its neighbors this is trivial. Then for each block an arbitrary number of points can be specified in the two local coordinate directions. This, however, results in blocks with discontinuous grid lines at the boundaries. We prefer instead to work with patched blocks with coincident grid lines. This means that the number of points on each edge, under the constrains described below, have to be computed. This is, for an arbitrary block topology, a non-trivial task. After some attempts to do this interactively, we decided instead to formulate the problem in a mathematical framework, as a optimization problem, and solve it by a well known technique.

To start with, let n_i denote the number of points on edge number i. For each block the following equations hold:

$$\begin{aligned} \textstyle\sum_{i\in E_{1,b}}(n_i-1) &= \textstyle\sum_{i\in E_{2,b}}(n_i-1) \\ \textstyle\sum_{i\in E_{3,b}}(n_i-1) &= \textstyle\sum_{i\in E_{4,b}}(n_i-1) \end{aligned} \tag{6}$$

where $E_{k,b}$ is the set of edges belonging to side k in block b. These are the compatibility equations which guarantee the same number of grid points on the corresponding block sides 1, 2 and 3, 4. For a solid wall edge n_i is governed by the number of points, N_i, in the geometry description. To ensure that n_i is not less than N_i we impose the constraint $N_i \leq n_i$. For the remaining edges we need lower and upper bounds, L_i and U_i on n_i i.e. $L_i \leq n_i \leq U_i$. The numbers L_i and U_i can be specified directly by the user or generated from a given grid point density function. The problem is now to find a solution to the underdetermined system (6) with the constraints above. Among all feasible solutions we want a solution which does not lead to unnecessary many grid points. Hence, it is reasonable to look for the solution which minimizes the total sum (or a weighted sum) of grid points. The final mathematical formulation then reads:

$$\begin{aligned} &\min_{n_i} \textstyle\sum_{i\in E} n_i \\ &\textstyle\sum_{i\in E_{1,b}}(n_i-1) = \textstyle\sum_{i\in E_{2,b}}(n_i-1) \\ &\textstyle\sum_{i\in E_{3,b}}(n_i-1) = \textstyle\sum_{i\in E_{4,b}}(n_i-1) \quad \text{for all blocks} \\ &n_i \geq N_i \quad i \quad \text{solid wall face,} \\ &L_i \leq n_i \leq U_i \quad i \quad \text{not solid wall face,} \quad n_i \quad \textit{integer} \end{aligned}$$

This is a linear integer programming problem which is solved by a standard technique based on branch and bounds.

If a global smoothing of the mesh is desired a search for singular points is also necessary. All the singular nodes are stored in an array together with their neighbor points and treated in a special way by the smoother as described in the next section. The block data structure used by the elliptic smoother is the same as for a multi-block flow solver which means that flow calculations can be done immediately when the mesh and the block data are available.

Grid Generation

The output from the mesh generator is a structured grid in each block, with coincident grid lines on the interfaces between two blocks. As mentioned in the previous section, an alternative approach, not considered in this work, is to allow the number and position of these grid points to vary independently in each block. This technique may prevent grid spreading, but needs a more sophisticated flow solver which includes conservative interpolation routines.

The initial mesh is generated by linear or hermitian transfinite interpolation. This gives often a good start grid due to the feature of the topology generator which provides blocks with rectangular shape and without strong curvature. The mesh quality is then further improved. The mesh is first stretched in the direction normal to solid wall boundaries. The resulting mesh will then better resolve the flow in these areas. The mesh is then smoothed by solving the well known elliptic equations (cf. Thompson et al. [11]):

$$\begin{cases} \alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta} & = -J^2 \left(x_\xi P + x_\eta Q \right) \\ \alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta} & = -J^2 \left(y_\xi P + y_\eta Q \right) \end{cases} \tag{7}$$

where (x, y) and (ξ, η) are coordinates in the physical and computational plane respectively, $\alpha = x_\eta^2 + y_\eta^2$, $\beta = x_\xi x_\eta + y_\xi y_\eta$, $\gamma = x_\xi^2 + y_\xi^2$ and $J = x_\xi y_\eta - x_\eta y_\xi$.

Following the technique suggested by Thomas and Middlecoff [10], the source terms P and Q are computed on the block boundaries and extended to the interior by linear interpolation.

$$\begin{cases} P = -\left(x_\xi x_{\xi\xi} + y_\xi y_{\xi\xi} \right) / \left(x_\xi^2 + y_\xi^2 \right) & \text{along the boundaries} \quad \eta = \text{constant} \\ Q = -\left(x_\eta x_{\eta\eta} + y_\eta y_{\eta\eta} \right) / \left(x_\eta^2 + y_\eta^2 \right) & \text{along the boundaries} \quad \xi = \text{constant} \end{cases} \tag{8}$$

Hence, the distribution of the boundary points spreads into the interior of the block. Finally, system (7) is discretized and solved iteratively by means of the Jacobi method. We are mainly interested in applying this technique in order to smooth the grid. If, however, there is a need for solving the equations completely, a multi-grid technique can be applied as an option, in order to speed up the convergence.

A special smoothing procedure is used for the singular points. Here, the mesh is smoothed iteratively by a discrete version of the undivided Laplacian operator:

$$\vec{r}_k^{n+1} = \vec{r}_k^{n} - \omega \sum_{i=1}^{N_p} (\vec{r}_k - \vec{r}_i)$$

where superscript n denotes the iteration counter, $\vec{r}_k$ is the singular point, $\vec{r}_i$, $i = 1, ..., N_p$ its neighbors and ω a relaxation parameter $(0 \leq \omega \leq 1)$.

RESULTS - COMPARISON OF METHODS

The two algorithms for the automatic generation of multi-block topologys are presented with the help of four different configurations. The generic configuration of a collection of four circles of different radius provides a first challenging test for the second of our methods. In Fig. 12 the global topology with 16 blocks and the final mesh are shown. The overall grid is characterized by a few non-smooth cells at the intersection of the blocks. Here singular points are created, which require special attention by the flow solver algorithm.

The double profile represents the first test case for method 1. Fig. 13 shows the global mesh topology consisting of 36 blocks. The overall topology type for this airfoil-flap configuration is a C-mesh, which is fairly well followed by the block structure. A close-up of this block topology together with the final mesh is given in Fig. 14. We note the smooth distribution of grid points (though the overall mesh is very coarse).

Two-dimensional mesh generation can also be used for axi-symmetric problems. As an example, we have meshed a valve-cylinder assembly which represents the first case to which both of our methods are applied. The blocks generated for this simplified internal flow configuration are depicted in Fig. 15. Method 1 generates nine original blocks (where some of them could be merged afterwards), while method 2 creates six blocks. While the shapes of the blocks in the valve part are similar, their topology in the cylinder part differs substantially. This difference, although not crucial, is transmitted to the meshes (Fig. 16).

A high-lift multi-element airfoil configuration represents the final test case. This geometry is the most difficult among our configurations due to the presence of several different length scales as well as strong curvature on the boundary. The present configuration is characterized by a main profile, a slat and two flaps that render a structured single-block generation quite impossible. The global C-type block topology for method 1 (75 blocks) and method 2 (21 blocks) is displayed in Fig. 17. The two approaches give substantially different arrangements of the blocks. With the divide-and-conquer method large blocks are generated that can extend from the surface of the airfoils to the far-field boundary of the domain. These large blocks are not obtained with the AFT-algorithm, which creates by far more blocks (where, again, some of them can be merged afterwards). The global meshes are displayed in Fig. 18. The underlying block topology becomes evident. Again, as with the generic circle configuration, with method 2 singular points can occur at block intersections. For a better look at the geometry a close-up of these block distributions is given in Fig. 19. We note the good clustering of a small number of blocks for method 2 in the region close to the airfoils. A similar view in Fig. 20 shows the computational grids. Finally, in Fig. 21 the solution of the Euler equations using the cell-centered multi-block solver of [4] on the mesh generated by method 2 is given. The figure shows the computed Mach number field for a free-stream Mach number of 0.15 and 10° angle of attack.

CONCLUDING REMARKS

Two different approaches for the automatic generation of the block topology of structured multi-block grids are presented. Both methods have been successfully applied to different geometries.

The first approach is based on an advancing front method. Compared to the original algorithm for triangles, the present method for the generation of rectangular elements is more complicated, partly because of attempts made to ensure robustness and the prevention of distorted elements. The additional computational costs are easily justifiable, since much less elements are created than for the generation of a completely unstructured mesh.

One of the major criticism leveled at the AFT is the background grid, which prevents the advancing front technique from being entirely automatic. A certain amount of user experience (or trial and error labour) is required to first place the vertices of the background grid triangles at the right locations and next to allocate suitable stretching parameters to these nodes. In our opinion the background grid needs to be replaced by a more automatic method, which provides the information about the element size. This information could for example be obtained from the curvature of the surface description. Further, robustness of the algorithm is not always guaranteed, due to the heuristics involved in keeping the algorithm stable and to enable its general application for a wide a range of configurations. Blocks need to be merged afterwards, both in order to reduce their total number and to avoid extrem differences in the block size. It is desirable to have a better control over the topology type of the global grid.

The second of our procedures is based on a divide-and-conquer principal. The algorithm basically consists of first flagging boundary nodes as *cut not permitted* (i.e. a corner), *cut required* or *cut permitted*, and then chosing an optimal cut between two nodes. The success of the algorithm depends on the definition of the function measuring the quality a cut. Obviously, a number of adequate functions can be constructed, and we do not claim having found the best one. It is however our experience that this function has to take into account the angles of the intersection of the cut with the boundary, the distance from the cut to the boundaries, the number of corners in the created subdomains, as well as the total number of boundary points still requiring a cut. The non-trivial problem of specifying the number of grid points on each block boundary can be formulated as a linear integer programming problem. With this approach, the only information that has to be supplied by the user is a lower bound for the number of points on a boundary edge.

A comparison of these two methods reveals better performances achieved with the divide-and-conquer approach. A smaller number of blocks is generated than with the AFT-method. The better control over the overall topology allows to cluster blocks in regions of complicated shapes, while at the same time only a few blocks are created away from the body surfaces. However, the problem of singular nodes still needs to be resolved. A future continuation of this work would therefore most likely be based on method 2. The basic divide-and-conquer principal applies directly to three-dimensional problems for realistic configurations. Modifications of the 2-D algorithms would comprise aspects such as the current way of flagging nodes and performing the cuts.

REFERENCES

1. Bergman, C.M.: *Developpement de Methodes Numeriques pour des Ecoulements Hypersoniques Non-visquex Autour d'Engins Spatiaux*, PhD thesis, Institut National Polytechnique de Toulouse, 1990.

2. Blacker, T.D. and Stephenson, M.B.: Paving: A New Approach to Automated Quadrilateral Mesh Generation, Intern. Journal for Num. Meth. in Engineering, 32, pp. 811-847, 1991.

3. Jenssen, C.B. and P. Weinerfelt: Automatic Multi-Block Mesh Generation in Two Dimensions, SINTEF

Industrial Mathematics, Report STF10 A93007, Trondheim, 1993.

4. Jenssen, C.B.: *Implicit Multi Block Euler and Navier-Stokes Calculations*, AIAA Journal, 32, No 9, pp. 1808–1814, 1994.

5. Löhner, R. and Parikh, P.: Generation of Three-Dimensional Unstructured Grids by the Advancing Front Method, AIAA-Paper 88-0515, Reno, January 1988.

6. Peraire, J. et al.: Adaptive Remeshing for Compressible Flow Computations, Journal of Comp. Phys. 72, pp. 449-466, 1987.

7. Schönfeld, T. and P. Weinerfelt, P.: The Automatic Generation of Quadrilateral Multi-Block Grids by the Advancing Front Technique, in Proceedings of the *Third International Conference on Numerical Grid Generation* in Barcelona, pp. 743-754, A. S.-Arcilla, J. Häuser, P.R. Eiseman, and J.F. Thompson (Eds.), Elsevier Science Publishers, North Holland, June 1991.

8. Steinthorsson, E. and Ameri, A.A.: Computations of Viscous Flows in Complex Geometries using Multiblock Grid Systems, AIAA-Paper 95-0177, Reno, January 1995.

9. Talbert, J.A. and Parkinsson, A.R.: Development of an Automatic Two-Dimensional Finite Element Mesh Generator using Quadrilateral Elements and Bezier Curve Boundary Definition, in *International Journal for Numerical Methods in Engineering*, 29, 1990.

10. Thomas,P.D. and Middlecoff, J.F.: Direct Control of the Grid Point Distribution in Meshes Generated by Elliptic Equations, AIAA Journal, 18, 1980.

11. Thompson, J.F., Warsi, Z.U.A. and Mastin, C.W.: *Numerical Grid Generation: Foundations and Applications*, North-Holland, 1985.

FIGURES

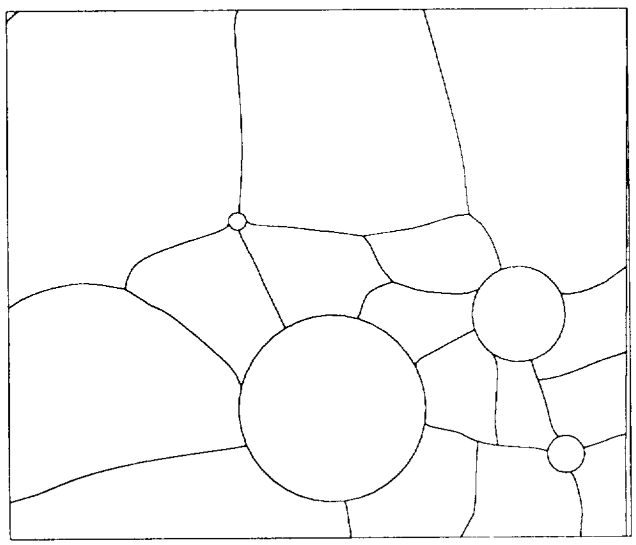
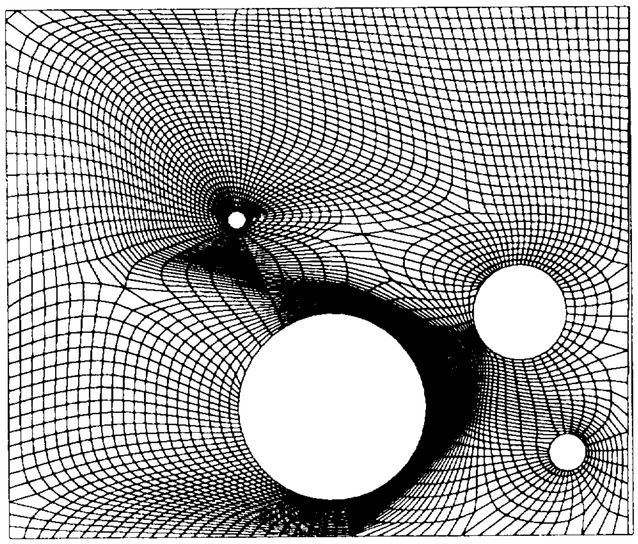

Figure 12: Global block topology (left) and final mesh for 4-circle configuration (method 2).

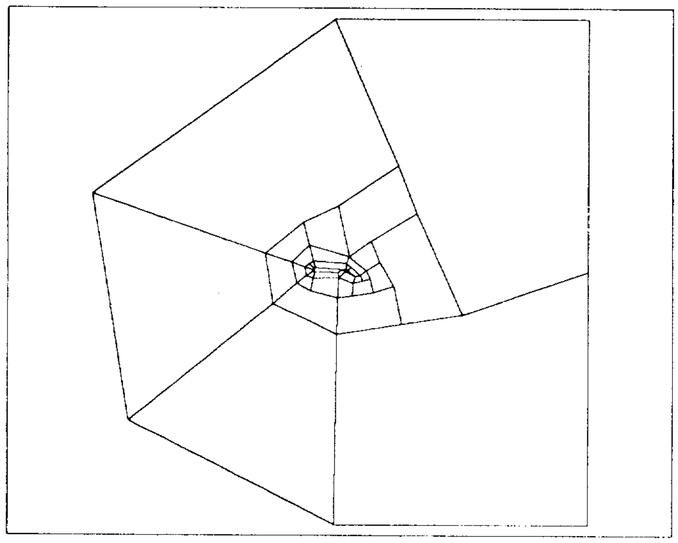

Figure 13: Global topology with 36 blocks for double profile (method 1).

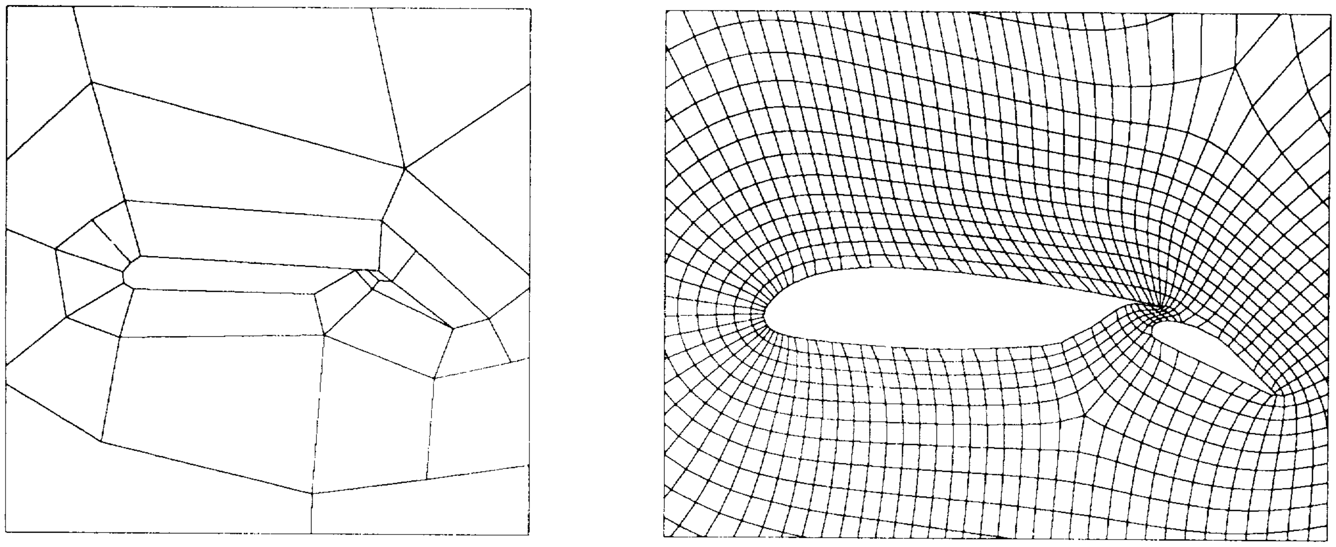

Figure 14: Close-up of block topology (left) and final mesh for double profile (method 1).

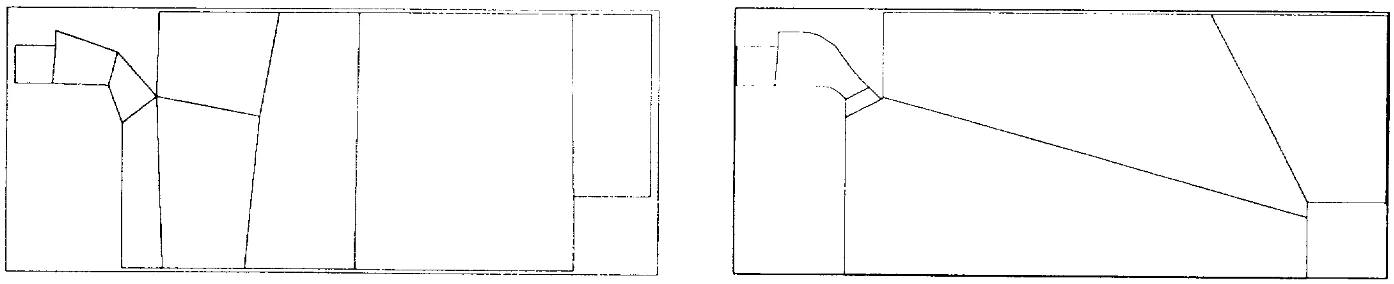

Figure 15: Valve-cylinder: global block topology for method 1 (left) and method 2.

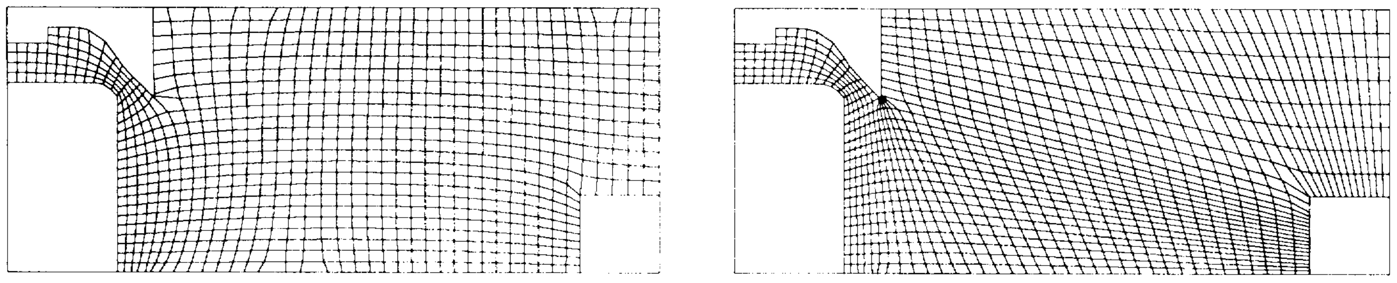

Figure 16: Valve-cylinder: final mesh for method 1 (left) and method 2.

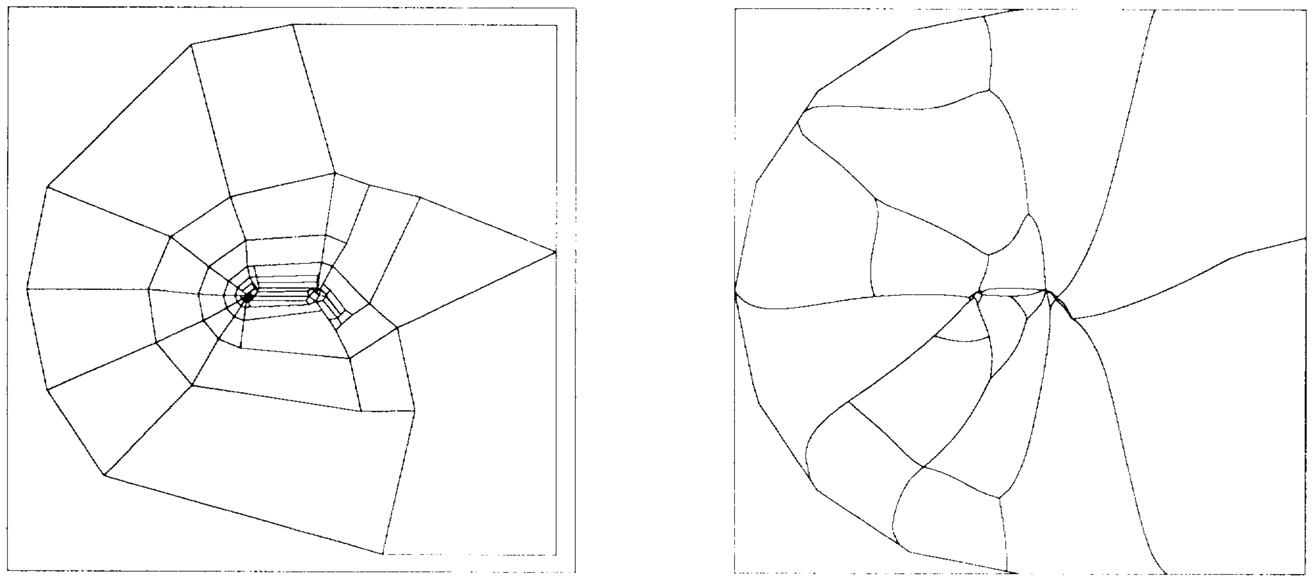

Figure 17: Multi-element airfoil: global block topology for method 1 (left) and method 2.

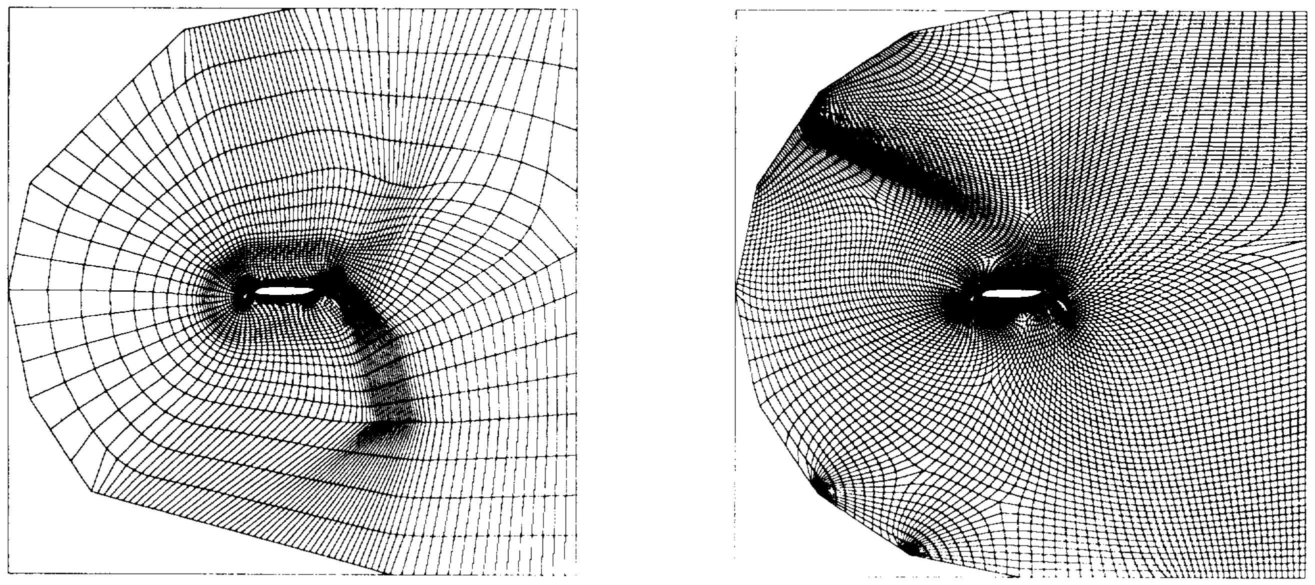

Figure 18: Multi-element airfoil: global grid for method (left) and method 2.

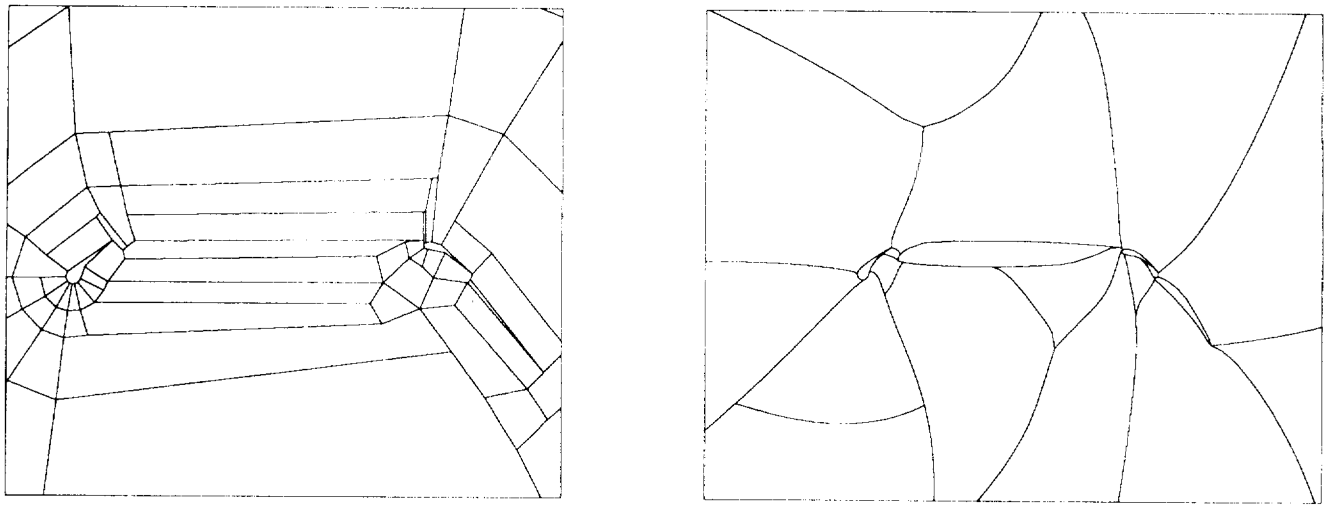
Figure 19: Multi-element airfoil: close-up of block topology for method 1 (left) and method 2.

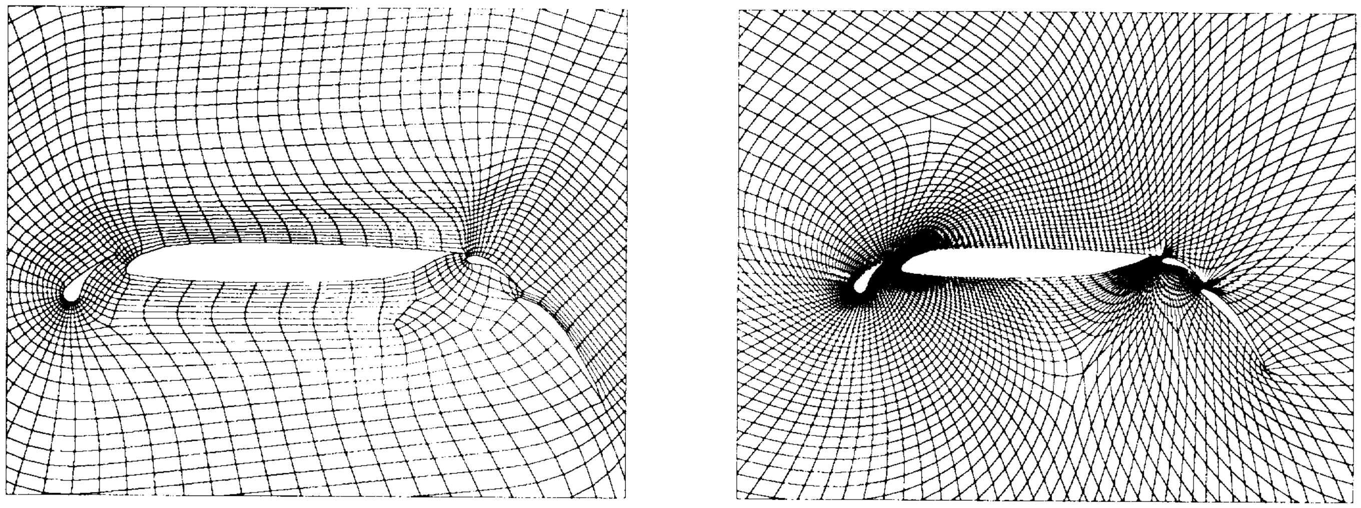
Figure 20: Multi-element airfoil: close-up of mesh for method 1 (left) and method 2.

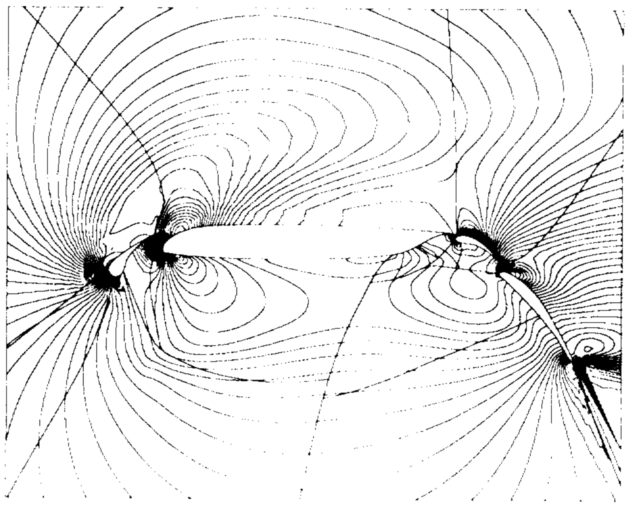
Figure 21: Multi-element airfoil: Mach number field for method 2.