

DOCUMENT CONTROL SHEET

	ORIGINATOR'S REF. NLR TP 96338 U	SECURITY CLASS. Unclassified																		
ORIGINATOR National Aerospace Laboratory NLR, Amsterdam, The Netherlands																				
TITLE Multiblock grid generation Part II: Multiblock aspects																				
PRESENTED AT 27th Computational Fluid Dynamics Course, at the Von Karman Institute (VKI) for fluid dynamics, Belgium, 25-29 March 1996.																				
AUTHORS S.P. Spekreijse, J.W. Boerstoel	DATE 960523	pp ref 51 20																		
DESCRIPTORS <table style="width: 100%; border: none;"> <tr> <td style="width: 33%;">Algorithms</td> <td style="width: 33%;">Hexahedrons</td> <td style="width: 33%;">Topology</td> </tr> <tr> <td>Body-wing configurations</td> <td>Interpolation</td> <td></td> </tr> <tr> <td>Computational geometry</td> <td>Multiblock grids</td> <td></td> </tr> <tr> <td>Computational fluid dynamics</td> <td>Navier-Stokes equation</td> <td></td> </tr> <tr> <td>Domain decomposition*</td> <td>Shape functions</td> <td></td> </tr> <tr> <td>Grid generation (mathematics)</td> <td>Structured grids (mathematics)</td> <td></td> </tr> </table>			Algorithms	Hexahedrons	Topology	Body-wing configurations	Interpolation		Computational geometry	Multiblock grids		Computational fluid dynamics	Navier-Stokes equation		Domain decomposition*	Shape functions		Grid generation (mathematics)	Structured grids (mathematics)	
Algorithms	Hexahedrons	Topology																		
Body-wing configurations	Interpolation																			
Computational geometry	Multiblock grids																			
Computational fluid dynamics	Navier-Stokes equation																			
Domain decomposition*	Shape functions																			
Grid generation (mathematics)	Structured grids (mathematics)																			
ABSTRACT Block-structured-grid-based methods offer a viable choice for solving viscous flows over complex aerodynamic configurations. Body-fitted structured grids are well suited for resolving the thin viscous layers developing in the vicinity of solid surfaces at high Reynolds numbers typically encountered in flight. A state-of-the-art structured flow solver, like ENSOLV, is very efficient in computing aerodynamic flows in the presence of such embedded boundary layers. An important step in routine application of structured-grid methodology to CFD applications is the grid generation process. At NLR and Fokker this is done with ENDOMO and ENGRID. The design of the domain decomposer ENDOMO and grid generator ENGRID has been realized in close relationship between an NLR CFD team as supplier and aerodynamic design and research teams from Fokker and NLR as customer. The partnership between the CFD development team and the design teams has led to user-friendly GUI based on concepts that automate much of the low level grid generation tasks.																				

NLR TECHNICAL REPORT

TP 96338 U

MULTIBLOCK GRID GENERATION

Part II: Multiblock aspects


by

S.P. Spekreijse and J.W. Boerstoel

This report has been presented at the 27th Computational Fluid Dynamics Course, at the Von Karman Institute (VKI) for fluid dynamics, Belgium, 25-29 March 1996.

This investigation has partly been carried out under a contract awarded by the Netherlands Agency for Aerospace Programs, contract number 01105N.

Division : Fluid Dynamics/Informatics

Prepared : SPS/  JWB/

Approved : FJH/  BO/ 



Completed : 960523

Order number : 526.004/101.123

Typ. : JvE



Contents

1	Introduction	5
2	Blocked flow domains	8
2.1	Topological elements, connectivity relations, and local coordinate systems	8
2.2	Degenerations	11
2.3	Geometrical shape of topological elements	12
2.4	Topology and Geometry file	15
3	Domain decomposition	15
3.1	Geometrical elements	16
3.2	Construction of topological elements	17
3.3	Refacing	17
3.4	Interactive domain decomposition	19
4	Grid generation	20
4.1	Grid dimension specification	21
4.2	Computational spaces	21
4.3	Grid generation in elementary edges	23
4.4	Grid generation in elementary faces	23
4.5	Grid generation in blocks	25
4.6	Local Grid refinement	25
4.7	Interactive grid generation	26
5	Applications	27
6	Summary	28



Appendices	29
A Piecewise cubic Hermite interpolation for curves	29
B Piecewise bicubic Hermite interpolation for surfaces	30
C Example of a topology file	32
References	34
57 Figures	

1 Introduction

For CFD computations of 3D flows, grids are required. The construction of such grids in 3D flow domains is nontrivial, when flow domain boundaries have complex shapes, like those of a complete aircraft. In such cases, CFD computations are today based on either multiblock structured grids, or on unstructured grids.

A multiblock grid may be described as a collection of blocks, with a grid in each block.

- Each *block* has the form of a deformed cube, and covers a part of the flow domain; a block has thus the topology of a unit cube, and has six block faces, twelve block edges, and eight block vertices, like the unit cube has. The blocks together should cover the flow domain completely. Moreover, a part of the block faces should cover the flow-domain boundaries completely. The covering of the flow domain by blocks may be arranged in various ways, see below for more details.

- Each block is subsequently covered by *hexahedral cells* of a grid. These grid cells should be packed cell-face-to-cell-face, and form a smooth, well-ordered, three-dimensional covering of the complete block. Each cell vertex defines a grid point. The collection of cell vertex points in a block defines the grid in the block. Because the grid cells in a block form a well-ordered three-dimensional covering of the block, these grid points may be stored in simple data structures, 3D arrays.

An unstructured grid may be described as a covering of the 3D flow domain by tetrahedral cells. Each cell has thus four cell faces, and four cell vertices. These tetrahedral cells are usually packed cell-face-to-cell-face, and should cover the flow domain completely. The collection of all cell vertex points defines the grid. Because here the grid cells are not required to cover the flow domain in a smooth, well-ordered way, the grid points cannot in general be stored in simple data structures like 3D arrays.

Multiblock grids can be classified in three different types, based on the way of covering of the flow domain by the blocks.

- In *overlapped (overset)* grids, adjacent blocks form 3D overlap regions, instead of e.g. interface surfaces. An overlapped region is common to at least two blocks.

- In *patched* grids, adjacent blocks form common interface surfaces. An interface surface lies in two block-face surfaces, one at either side of the interface surface. The two grids in the two blocks induce in general two different subgrids in the common interface surface. When this is true, the grid is called here a patched grid.

- In C^0 -*continuous* grids, adjacent blocks with a common interface surface have also a common subgrid in that interface surface.

Overlapped grids, patched grids, and C^0 -continuous grids form thus a hierarchy of multiblock structured grids of increasing structure. The covering of the flow domain by blocks is, with overlapped grids, in principle completely unstructured. This is, to a lesser extent, also the case for patched and C^0 -continuous grids. C^0 -continuous grids require extra information to allow a coupling of the two grids in two adjacent blocks such that in common interface surfaces a common subgrid is defined.

The advantages and disadvantages of unstructured grids and the various kinds of multiblock structured grids with respect to each other should be considered from all viewpoints which are relevant for industrial applications of CFD technology.

- **Geometry manipulation.** Complex surface shapes of 3D flow-domain boundaries, like complete transport and fighter aircraft, can in fact only be efficiently processed for CFD work, if accurate, flexible, and reliable surface-representation and -manipulation methods are used.

This will be made clear in this report. For CFD-oriented surface-processing activities in grid generators and flow solvers, CAD/CAM packages are not so useful, because they have a high user-threshold for CFD specialists.

- **Grid generation.** Unstructured 3D grids require the smallest manhour effort for their generation, and C^0 continuous grids the most.
- **Flow solvers.** C^0 continuous grids are the simplest and most convenient to deal with. With C^0 continuous grids it is easy to guarantee conservation across block interfaces, achieve high numerical accuracy, and to apply efficient solution methods. Unstructured grids offer in these respects still problems.
- **Postprocessing.** The more structure in the grids, the easier the graphical interactive inspection of smoothness and accuracy of grids and numerical flow solutions.

Multiblock grids may be considered to provide a fair compromise between the advantages and disadvantages of structured and unstructured grids. The structured grids within the blocks allow high computational efficiency and numerical accuracy of computation results, while the unstructuredness of block coverings of flow domains allows the treatment of complex configurations. These advantages of multiblock grids come with a price of increased manhour effort (compared to the situation about ten years ago, when monoblock grids were still usual).

The successful use of block-structured-grid-based methods for solving viscous flows over complex aerodynamic configurations is demonstrated in recent conferences on grid generation [1],[2],[3],[4], [5],[6]. Almost all aerospace organizations involved in CFD simulations have some type of in-house multiblock grid generation capability. This suggests that the manhour effort for multiblock grid generation is generally considered acceptable.

In this paper we will mainly consider only C^0 -continuous grids. Only a restrictive type of patched grids, based on local grid refinement of the structured grids in the blocks, will also be briefly described. For such restrictive patched grids it is also possible to preserve conservation across block interfaces [9].

The generation of a C^0 continuous grid involves three major subsequent processes:

- a) surface modeling,
- b) domain decomposition i.e. the construction of the blocks within the flow domain and
- c) grid generation i.e. the construction of the structured grids within each block.

Grid generation processes start thus with surface modeling of flow boundaries with, in general, CAD/CAM software. The purpose of surface modeling is to obtain the geometric definition of configuration surfaces in a form which is acceptable for CFD work. Often, configuration surfaces are provided with a computer file which was generated by a Computer-Aided Design CAD system (like CATIA). Typical standard forms of such files are IGES and VDA files. CAD geometries have usually to be edited and/or trimmed, to correct geometric defects which are not acceptable in CFD work (i.e. gaps, or non-physical edges of surface elements), or to modify the true geometry for the purpose of CFD analysis (i.e. remove parts of the geometry that are not relevant for CFD analysis). Sometimes, surface definitions of only the individual components of a configuration (e.g. wing, fuselage, pylon, nacelle) are provided as input of CFD computations. The intersections of the various components have then to be determined.

Surface modeling for CFD work may end with a geometric definition of the configuration surface consisting of surface patches, each surface patch being pointwise defined by a 2D array of points. These pointwise defined surface patches should abut cleanly, with no gaps, overlaps, doubly defined regions, or non-physical protrusions. The creation of these surface patches tends to be a time-consuming effort, requiring mastery of large, complex CAD-based



systems like ICEM/CFD or Unigraphics. However, such CAD-based systems are further well-equipped for this kind of surface-modeling work.

After completion of the surface modeling task, the domain-decomposition task has to be executed. The physical flow domain is then subdivided into an unstructured collection of blocks. Below, only decompositions with non-overlapping blocks will be considered. Domain decomposition is an highly interactive task. An advanced Graphical User Interface (GUI) on a good workstation are needed to support a user during domain decomposition. The GUI should provide a user with a set of efficient functions, to facilitate the domain decomposition process. Nowadays, the interactive generation of blocks is still the most time consuming and difficult part of the multiblock grid generation process.

After domain decomposition, a blocked flow domain is available, and the grid generation step can be executed, i.e. the construction of a structured grid in each block. Grid generation is also a highly interactive task, but is much more simple and less time consuming than domain decomposition. The result of the grid generation is a multiblock grid, which may be used in a flow solver to compute solutions of Euler and/or Navier-Stokes equations.

A complete set of mutually tuned codes, for surface modeling, domain decomposition, grid generation and flow calculation, is called a flow simulation system. Such a flow simulation system has been developed at NLR for the computation of flows about complete aircraft configurations, including propulsion-system components. The system is known as the ENFLOW (Euler/Navier-Stokes FLOW) system [7],[8]. Fig.1 shows the layout of the system and summarizes its use for CFD work. The commercial ICEM/CFD code is used for CAD/CAM type surface modeling. The subdivision of a three dimensional flow domain into blocks is done with the graphical interactive domain modeler ENDOMO. The graphical interactive grid generator ENGRID is used for the computation of structured grids in the blocks. Given a multiblock grid, the flow solver ENSOLV computes the solution of the Euler and/or Navier-Stokes equations with respect to specified boundary conditions. The system contains further about 20 auxiliary codes for all kinds of tasks which makes the life of CFD specialists easier (visualization codes, file conversion codes, etc.). The ENFLOW system has been used for industrial flow calculations around many kinds of configurations: various types of transport aircraft configurations, various fighter configurations, space shuttles, wing bodies, supersonic projectiles, etc. In the near future, the ENFLOW system will be extended with grid adaptation of multiblock structured grids based on mesh movement [15],[16].

From the description above it should be apparent that multiblock grid generation with the ENFLOW system is a user-in-the-loop task. Therefore, one of the goals of the software design of ENDOMO and ENGRID has been to automate as much as possible, so that the user may concentrate on handling of the topology of block decompositions and on providing sufficient grid quality. This paper will describe some of the most important automation tools that are used in the ENDOMO and ENGRID codes. An excellent general overview about multiblock grids and its use in CFD can be found in [10] and [11].

The paper is organized as follows. Section 2: Theory for blocked flow domains is described. The topological elements of block decompositions and their connectivity relations are defined. Compound edges and compound faces are introduced; these allow block decompositions to be more or less unstructured collections of blocks. Degenerations of topological elements are considered. The design of geometrical shape functions of topological elements is described. Section 3: The domain-decomposition process is described. In ENDOMO, there is a clear distinction between topological and geometrical elements. The user constructs geometrical elements which are in turn used to build the topological elements. Section 4: Grid generation

with ENGRID for the construction of structured grids in blocks is described. Section 5: Some applications are given. Section 6: The most important multiblock concepts and ideas, that are used in the software design of ENDOMO and ENGRID are summarized.

2 Blocked flow domains

2.1 Topological elements, connectivity relations, and local coordinate systems

A blocked flow domain consists of topological elements. These elements are blocks, faces, edges and vertices. The elements are topologically equivalent to, respectively, a unit cube, unit square, unit interval, and point. These topological equivalences are the basis for the construction of well-structured grids in each element.

A block has six faces, twelve edges, and eight vertices as topological boundary elements. A face has four edges and four vertices as topological boundary elements. An edge has two vertices as topological boundary elements.

Blocks, faces, edges and vertices are identified by labels B , F , E , V , respectively, from sets $\{B\}$, $\{F\}$, $\{E\}$, and $\{V\}$, which are so-called "label sets". In practice, the labels are integers in \mathcal{N} : the set of positive natural numbers. For example, B_{112} is block number 112, F_{63} is face number 63, etc., and, in computer codes, only the integer subscripts are used to identify block B_{112} and face F_{63} .

The way how of blocks, faces, edges, and vertices are connected to each other over common boundary elements defines the topology of a blocked flow domain. It appears that only five types of connectivity relations between the above topological elements are sufficient to define the complete topology of a blocked flow domain.

The first type concern block-to-face relations. Each block B is connected to its six faces, and this is expressed by a mapping of the form

$$\forall B \in \{B\} : B \mapsto (F_1, F_2, F_3, F_4, F_5, F_6). \quad (1)$$

For reasons made clear below, the face labels are assumed to be ordered in pairs: faces F_1 and F_2 should be in block B opposite to each other, faces F_3 and F_4 should also be opposite to each other, and faces F_5 and F_6 are the remaining pair of opposite faces in block B .

For example, in a computer code, a relation defined by Eq.(1) could have the form

$$B_{112} = [23 \ 44 \ 2 \ 12 \ 9 \ 212],$$

meaning that block B_{112} has the faces F_{23} , F_{44} , F_2 , F_{12} , F_9 , and F_{212} as opposite pairs of faces.

Two blocks will be connected to each other block-face-to-block-face, if they have a common face label in their block-to-face relations.

The second type of connectivity relations concern face-to-edge connectivity relations. Each face F is connected to its four edges, and this is expressed by a mapping of the form

$$\forall F \in \{F\} : F \mapsto (E_1, E_2, E_3, E_4). \quad (2)$$

Here, too, the edge labels are assumed to be ordered in pairs: edges E_1 and E_2 should be in face F opposite to each other, and the two remaining edges E_3 and E_4 are then also opposite to each other.

Two faces will be connected to each other face-edge-to-face-edge, if they have a common edge label in their face-to-edge relations.

The third type of connectivity relations concern edge-to-vertex relations. Each edge E is related to its two vertices, and this is expressed by a mapping of the form

$$\forall E \in \{E\} : E \mapsto (V_1, V_2). \quad (3)$$

The two vertex labels are here also to be considered ordered as a pair of opposite vertices, for reasons explained below.

Two edges will be connected to each other edge-vertex-to-edge-vertex, if they have a common vertex in their edge-to-vertex relations.

The topological equivalences between blocks, faces, edges and vertices with, respectively, unit cubes, unit squares, unit intervals, and points, have as a consequence that the topological boundary elements of each block, face, and edge must be connected to each as in the unit cube, square, and interval. These properties should be (automatically!) built in the above connectivity mappings during domain decomposition work.

The connectivity relations are such that each face has exactly four corner-vertices. Similarly, each block has exactly eight corner-vertices and twelve boundary edges. The four corner-vertices of a face, and the eight corner-vertices and twelve boundary-edges of a block can be (completely automatically!) computed from the above connectivity relations Eqs.(1), (2), (3), and lead to so-called derived connectivity relations.

The four corner-vertices of each face are then given by a derived connectivity relation of the form:

$$\forall F \in \{F\} : F \mapsto (V_1, V_2, V_3, V_4). \quad (4)$$

The topology of a face is shown in Fig. 3. In this example, vertex V_1 is defined as the intersection vertex of edge E_1 and edge E_3 , thus as the common vertex label of the two connectivity mappings of these edges.

The eight corner-vertices of a block are given by derived connectivity relations of the form:

$$\forall B \in \{B\} : B \mapsto (V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8). \quad (5)$$

The twelve boundary-edges of a block are given by derived connectivity relations of the form:

$$\forall B \in \{B\} : B \mapsto (E_1, E_2, E_3, E_4, E_5, E_6, E_7, E_8, E_9, E_{10}, E_{11}, E_{12}). \quad (6)$$

The topology of a block is shown in Fig. 2.

The three types of connectivity relations specified by Eqs.(1), (2), and (3) are sufficient to describe the complete topology of block coverings of flow domains, in which the blocks are packed block-face-to-block-face.

In order to allow more freedom in block coverings of flow domains, compound faces and compound edges are introduced. These compound elements should allow partial block-boundary interfacing. A face of a block can then be adjacent to more than one other block [14]. This allows unstructuredness of block coverings of flow domains, and leads, compared to block-face-to-block-face coverings, to much fewer and larger blocks. An illustration of this effect is shown in Figs. 4, 5, 6 which is a simple 2D example of a block structured grid for a part of the harbour of Rotterdam [17]. The use of compound elements also enhances the flexibility of the domain decomposition process.



A compound face is defined as consisting of two subfaces that are joined together at a common edge (see Fig. 13). Each subface of a compound face is also allowed to be compound again. A face which is not compound is elementary. The set of compound and elementary faces are denoted as $\{F^c\}$ and $\{F^e\}$. Thus $\{F\} = \{F^e\} \cup \{F^c\}$.

The fourth kind of connectivity relations mentioned at the beginning of this section are compound-face-to-subfaces mappings. For each compound face F there is a mapping of the form

$$\forall F \in \{F^c\} : F \mapsto (F_1, F_2). \quad (7)$$

giving the two subfaces F_1 and F_2 of the compound face.

The fifth and last kind of connectivity relations are compound-edge-to-subedges relations. A compound edge consists of two subedges, that are joined together at a common vertex (see Fig. 12). The set of compound and elementary edges are denoted as $\{E^c\}$ and $\{E^e\}$. Thus $\{E\} = \{E^e\} \cup \{E^c\}$, and the fifth kind of connectivity relations are of the form

$$\forall E \in \{E^c\} : E \mapsto (E_1, E_2). \quad (8)$$

The derived connectivity relations of compound edges and compound faces must also obey certain restrictions. For example, the two subfaces of a compound face, given by Eq.(7), must have a unique common edge with one edge label. Similarly, the two sub-edges of a compound edge, given by Eq.(8), must have a unique common vertex with one vertex label.

In a multiblock system, each topological element has its own local coordinate system. The local computational coordinate system in a topological element is an example of such a local coordinate system. But also the parametrization of the curve of an edge or the surface of a face define a local coordinate system in the edge and face. The unit cube for blocks, the unit square for face and the unit interval for edges can always be used as the range of the local coordinates. For example, the normalized arclength $u \in [0, 1]$ can be used as parametrization of the curve of an edge. The computational coordinate $\xi \in [0, 1]$ of an edge can be defined as $\xi_i = i/N \mid i = 0 \dots N$ where N is the number of grid cells on the edge. The computational space of a face can be defined as $(\xi, \eta) \in [0, 1]^2$ with $(\xi_{ij}, \eta_{ij}) = (i/N, j/M) \mid i = 0 \dots N, j = 0 \dots M$ where N and M are the number of grid cells in the two computational directions. The parametrization of the surface of a face can always be defined as $\vec{x}_F : (u, v) \in [0, 1]^2 \mapsto \mathcal{R}^3$ where u and v are defined as normalized arclength along the four boundaries of the surface of the face. The computational space of a block can be defined as $(\xi_{ijk}, \eta_{ijk}, \zeta_{ijk}) = (i/N, j/M, k/L) \mid i = 0 \dots N, j = 0 \dots M, k = 0 \dots L$ where N, M and L are the number of grid cells in the three coordinate directions.

The topological connectivity relations are used to define the orientation of a local coordinate system in a topological element. For a block B , with $B \mapsto (F_1, F_2, F_3, F_4, F_5, F_6)$, the orientation of the local coordinate system $(\xi, \eta, \zeta) \in [0, 1]^3$ is such that $\xi = 0$ at face F_1 , $\xi = 1$ at face F_2 , $\eta = 0$ at face F_3 , $\eta = 1$ at face F_4 , $\zeta = 0$ at face F_5 , $\zeta = 1$ at face F_6 (see Fig. 2). For a face F , with $F \mapsto (E_1, E_2, E_3, E_4)$, the orientation of the local coordinate system $(\xi, \eta) \in [0, 1]^2$ is such that $\xi = 0$ at edge E_1 , $\xi = 1$ at edge E_2 , $\eta = 0$ at edge E_3 , $\eta = 1$ at edge E_4 (see Fig. 3). For an edge E , with $E \mapsto (V_1, V_2)$, the orientation of the local coordinate system $\xi \in [0, 1]$ is such that $\xi = 0$ at vertex V_1 , $\xi = 1$ at vertex V_2 .

Local coordinates are very useful to completely automatically relate to each other functions of different topological elements, which must satisfy nontrivial continuity requirements. This occurs for example during grid generation when a generated grid in an edge (face) must be stored in all blocks to which the edge (face) belongs (see Section 4.2). It is also needed for



the construction of the geometrical shape function of a face for which it is required that the surface of the face exactly fits the four curves of the face-edges (see Section 2.3).

2.2 Degenerations

During domain decomposition, it is sometimes inevitable to use degenerated blocks (for example in a polar grid region where a block with a block-face degenerated to a point or curve is needed). Hence, degeneration of a topological element should be allowed. However there are restrictions for the types of degeneration that are allowed. These restrictions are caused by the following considerations.

In a multiblock grid, each topological element has its own structured grid. The orientation of the computational space is defined by the topological connectivity relations. For a block B , with $B \mapsto (F_1, F_2, F_3, F_4, F_5, F_6)$, the orientation of the computational space $(\xi_B, \eta_B, \zeta_B) \in [0, 1]^3$ is such that $\xi_B = 0$ at face F_1 , $\xi_B = 1$ at face F_2 , $\eta_B = 0$ at face F_3 , $\eta_B = 1$ at face F_4 , $\zeta_B = 0$ at face F_5 , $\zeta_B = 1$ at face F_6 (see Fig. 2). For a face F , with $F \mapsto (E_1, E_2, E_3, E_4)$, the orientation of the computational space $(\xi_F, \eta_F) \in [0, 1]^2$ is such that $\xi_F = 0$ at edge E_1 , $\xi_F = 1$ at edge E_2 , $\eta_F = 0$ at edge E_3 , $\eta_F = 1$ at edge E_4 (see Fig. 3). For an edge E , with $E \mapsto (V_1, V_2)$, the orientation of the computational space $\xi_E \in [0, 1]$ is such that $\xi_E = 0$ at vertex V_1 , $\xi_E = 1$ at vertex V_2 . The size of the computational space of a topological element, i.e. the number of grid cells in each computational coordinate direction, is defined during grid generation (see Section 4.1). For grid generation purposes, it is necessary that the computational space of a topological element is uniquely embedded in the computational space of a topological element to which it belongs (see Section 4.2). The embedding of computational spaces into each other depends on topology and the size of the computational spaces and is automatically determined when the grid dimensions are known. For example, Fig. 7 shows how the computational space of an edge and face may be embedded in the computational space of a block.

Degenerated topological elements are automatically identified by the topological connectivity relations. An elementary edge which is degenerated to a point is characterized by the connectivity relation $E \mapsto (V_1, V_2)$ with $V_1 = V_2$. Such an edge is called collapsed. The orientation of the computational space of a collapsed edge is undefined, but can be chosen arbitrarily (in a computer code) because all grid points along the edge will coincide with the position of the vertex. A closed-curve edge with equal vertices can not be used because the orientation is then not defined and can not be chosen arbitrarily. Only closed-edge curves with different vertices (although their geometrical position is the same) are allowed. Thus from a topological point of view, there are two types of elementary edges: collapsed and not-collapsed. Collapsed edges are also very special in the sense that the number of grid points along such an edge is not unique. This is illustrated in Fig.8 where edge E is a collapsed edge with vertex V . The number of grid points along edge E depends on the block to which the edge belongs and is not unique.

Degenerations of compound edges are not allowed. An elementary sub-edge of a compound edge is not allowed to be collapsed. The reason for this is that size and orientation of the computational space of a collapsed edge is not uniquely defined so that the embedding of the computational space of a collapsed sub-edge in the computational space of the compound edge is also not uniquely defined.

Allowed types of degenerated elementary faces are shown in Fig.9. An elementary face will be degenerated to a curve if two opposite edges are equal (i.e. $E_1 = E_2$ or $E_3 = E_4$



in Eq. (2)). In that case, one coordinate direction is undefined. An elementary face will be degenerated to a point if both pairs of opposite edges are equal. Then both coordinate directions are undefined

Allowed types of degenerated compound faces are illustrated in Fig.10. Not all possible kinds of degenerations are allowed. The rule is that the computational space of a subface is uniquely embedded in the computational space of the compound face.

Allowed types of degenerated blocks are illustrated in Fig.11. These types of degenerated blocks have in common that the orientation of the computational space of the blocks is always uniquely defined. For example, a block with two equal opposite faces is not allowed. This is also not needed because such a block would be degenerated to a quadrilateral surface.

2.3 Geometrical shape of topological elements

Another very important aspect of blocked flow domains are the geometrical shape functions of edges and faces. Each edge (face) is pointwise defined by an 1D (2D) array of control points in physical space and piecewise cubic Hermite interpolation is used to construct a smooth curve (surface) which is passing through the set of control points without introducing spurious oscillations (as may occur with standard cubic spline interpolation).

Sometimes the pointwise definition of a face does not have a point-to-point match with the pointwise definition of a face-edge. Correction functions are introduced so that this mismatch is automatically repaired. These correction functions are needed to guarantee that a multiblock grid, created with ENGRID, is C^0 continuous. The need for correction functions is often overlooked in the literature.

The geometrical shape of a vertex V is a point, given by $\vec{x}_V = (x, y, z)^T$. The geometrical shape of an edge is a curve segment. Consider an elementary edge E with vertices V_1 and V_2 given by the connectivity relation $E \mapsto (V_1, V_2)$. From a geometrical point of view, there are two types of elementary edges: default edges and non-default edges. A default elementary edge is an edge of which the edge-curve shape is a straight-line segment between the two edge-vertices. The geometrical shape function of an default elementary edge is given by

$$\vec{x}_E(u) = \vec{x}_{V_1}(1 - u) + \vec{x}_{V_2}u, \quad (9)$$

where $u \in [0, 1]$, V_1 and V_2 are the two vertices of edge E . Notice that u is the normalized arclength and $\vec{x}_E(0) = \vec{x}_{V_1}$ and $\vec{x}_E(1) = \vec{x}_{V_2}$. Also notice that the coordinate direction of u is defined by the topological connectivity relations and is thus the same as for the computational coordinate. This is a general rule which also holds for compound edges and elementary and compound faces.

A non-default elementary edge is an edge of which the edge-curve shape is described by an ordered one dimensional sequence of control points given by $\vec{x}_{P_1} \dots \vec{x}_{P_N}$. In computer codes, an ordering of the control points is used such that the first control point \vec{x}_{P_1} is closest to vertex V_1 and the last control point \vec{x}_{P_N} is closest to vertex V_2 . This is done automatically. Piecewise cubic Hermite interpolation between the control points is used to construct a smooth C^1 curve $\vec{x}_{cur} : u \in [0, 1] \mapsto \mathcal{R}^3$ which is passing through the set of control points with a geometrical shape as one would intuitively expect, without introducing spurious oscillations. Appendix A describes Hermite interpolation between control points. Normalized arclength is used for the coordinate u . Fig. 19 illustrates that Hermite interpolation gives a monotonic interpolation curve while cubic spline interpolation may easily introduce unwanted oscillations.



The first and last control point of a non-default edge may not match the edge-vertices exactly. It is required that the edge-curve is passing exactly through the edge-vertices; otherwise two edges which share the same vertex may not be continuously connected. This is obtained by defining the geometrical shape function of a non-default elementary edge as

$$\vec{x}_E(u) = \vec{x}_{cur}(u) + \vec{x}_{cor}(u), \quad (10)$$

where $\vec{x}_{cor}(u)$ is a correction function defined by

$$\vec{x}_{cor}(u) = (\vec{x}_{V_1} - \vec{x}_{P_1})H_0(u) + (\vec{x}_{V_2} - \vec{x}_{P_N})H_1(u), \quad (11)$$

where \vec{x}_{P_1} , \vec{x}_{P_N} are the positions of the first and last control point, and H_0 , H_1 are cubic Hermite functions defined in Eq.(41) below. The result of the correction function is that non-default edges also precisely obey that $\vec{x}_E(0) = \vec{x}_{V_1}$ and $\vec{x}_E(1) = \vec{x}_{V_2}$.

A compound edge can be considered as a binary tree structure, with elementary edges at the leaves and compound edges at the nodes. The geometrical shape function of a compound edge is recursively defined in terms of the geometrical shape functions of its sub-edges.

Consider a compound edge E with sub-edges E_1 and E_2 . Let $\vec{x}_{E_1}(u_1), \vec{x}_{E_2}(u_2)$ be the two geometrical shape functions of E_1 and E_2 . Assume that the two sub-edges E_1 and E_2 are situated in the compound edge E as shown in Fig. 12. Let L_1 and L_2 denote the length of edges E_1 and E_2 . Assume that u_1 and u_2 are normalized arclength. Thus $\|\frac{d}{du_1}\vec{x}_{E_1}(u_1)\| = L_1$, $\|\frac{d}{du_2}\vec{x}_{E_2}(u_2)\| = L_2$. Then $\vec{x}_E(u)$ is defined as

$$\vec{x}_E(u) = \begin{cases} \vec{x}_{E_1}(u_1) & \text{with } u = L_1u_1/(L_1 + L_2), u_1 \in [0, 1] \\ \vec{x}_{E_2}(u_2) & \text{with } u = (L_1 + L_2u_2)/(L_1 + L_2), u_2 \in [0, 1] \end{cases} \quad (12)$$

It is easily verified that $\vec{x}_E(u)$ is also arclength scaled, i.e. $\|\frac{d}{du}\vec{x}_E(u)\| = L_1 + L_2$. There are in fact eight possibilities, defined by the topology, how the two sub-edges E_1 and E_2 can be situated in the compound edge E . Fig. 14 show the eight possibilities. Each case has its own unique definition of $\vec{x}_E(u)$; Eq.12 is just an example of one of the eight possibilities.

The final result of this recursive definition is that a compound edge-curve can be considered as a composite curve consisting of L elementary edge-curves. The compound edge-curve is defined on L intervals $0 = u_0 < \dots < u_L = 1$. Each interval is mapped to an elementary edge-curve. The ratio of the size of two intervals is equal to the ratio of the length of the two corresponding elementary edge-curves. The relation between the global coordinate $u \in [0, 1]$ of the compound-edge curve and a local coordinate $\tilde{u} \in [0, 1]$ of an elementary edge-curve is defined by the position of the corresponding interval and the orientation of the elementary edge in the compound edge. Finally, notice that $\vec{x}_E(u)$ is continuous and C^1 in the interior of the intervals.

The geometrical shape of a face is a quadrilateral surface. From a geometrical point of view, there are two types of elementary faces: default faces and non-default faces. The geometrical shape of a default elementary face is defined as a bilinearly blended Coon's patch bounded by the four curves of the four face-edges. The geometrical shape function $\vec{x}_F : (u, v) \in [0, 1]^2 \mapsto \mathcal{R}^3$ is thus defined as

$$\vec{x}_F(u, v) = (1-u, u) \begin{pmatrix} \vec{x}_{E_1}(v) \\ \vec{x}_{E_2}(v) \end{pmatrix} + (1-v, v) \begin{pmatrix} \vec{x}_{E_3}(u) \\ \vec{x}_{E_4}(u) \end{pmatrix} - (1-u, u) \begin{pmatrix} \vec{x}_{V_1} & \vec{x}_{V_3} \\ \vec{x}_{V_2} & \vec{x}_{V_4} \end{pmatrix} \begin{pmatrix} 1-v \\ v \end{pmatrix}. \quad (13)$$



At the boundary, the geometrical shape function of a default face is equal to the geometrical shape function of an edge, for example $\vec{x}_F(0, v) = \vec{x}_{E_1}(v)$. In Eq.13 is assumed that the coordinate direction of the face edges correspond with the coordinate direction of the face itself. If this is not the case, for example for edge E_1 , then $\vec{x}_{E_1}(1 - v)$ must be used instead of $\vec{x}_{E_1}(v)$ in Eq.13. Notice that the geometrical shape function of a compound edge is needed in Eq.(13) if a face-edge is compound.

A non-default elementary face is a face of which the geometrical surface shape is described by a well-ordered two-dimensional sequence of control points. In Appendix B is described how piecewise bi-cubic Hermite interpolation between the control points is applied to construct a smooth C^1 surface $\vec{x}_{sur} : (u, v) \in [0, 1]^2 \mapsto \mathcal{R}^3$ which is passing through the set of control points with a geometrical shape as one would intuitively expect, without introducing spurious oscillations.

It is also required that the surface of a face should fit the curves of the four edges of the face exactly. Otherwise, two faces which share the same edge will not be continuously connected. This is obtained by defining the geometrical shape function of a non-default elementary face as

$$\vec{x}_F(u, v) = \vec{x}_{sur}(u, v) + \vec{x}_{cor}(u, v), \quad (14)$$

where $\vec{x}_{cor}(u, v)$ is a correction function defined by

$$\begin{aligned} \vec{x}_{cor}(u, v) = & (H_0(u), H_1(u)) \begin{pmatrix} \vec{x}_{E_1}(v) - \vec{x}_{sur}(0, v) \\ \vec{x}_{E_2}(v) - \vec{x}_{sur}(1, v) \end{pmatrix} + (H_0(v), H_1(v)) \begin{pmatrix} \vec{x}_{E_3}(u) - \vec{x}_{sur}(u, 0) \\ \vec{x}_{E_4}(u) - \vec{x}_{sur}(u, 1) \end{pmatrix} \\ & - (H_0(u), H_1(u)) \begin{pmatrix} \vec{x}_{V_1} - \vec{x}_{sur}(0, 0) & \vec{x}_{V_3} - \vec{x}_{sur}(0, 1) \\ \vec{x}_{V_2} - \vec{x}_{sur}(1, 0) & \vec{x}_{V_4} - \vec{x}_{sur}(1, 1) \end{pmatrix} \begin{pmatrix} H_0(v) \\ H_1(v) \end{pmatrix}. \end{aligned} \quad (15)$$

In this way, we also obtain that at the boundary, the geometrical shape function of a non-default face is equal to the geometrical shape function of an edge, for example $\vec{x}_F(0, v) = \vec{x}_{E_1}(v)$. In Eq.(15) is again assumed that the coordinate direction of the face edges correspond with the coordinate direction of the face itself. If this is not the case, for example for edge E_1 , then again $\vec{x}_{E_1}(1 - v)$ must be used instead of $\vec{x}_{E_1}(v)$ in Eq.15.

A compound face can be considered as a binary tree structure with elementary faces at the leaves and compound faces at the nodes. Therefore, it is sufficient to define the geometrical shape function of a compound face given the geometrical shape functions of the two sub-faces. In this way, the geometrical shape function of a compound face is recursively defined.

Consider a compound face F with sub-faces F_1 and F_2 . Let $\vec{x}_{F_1}(u_1, v_1), \vec{x}_{F_2}(u_2, v_2)$ be the two geometrical shape functions of F_1 and F_2 . Assume that $\vec{x}_{F_1}(u_1, v_1), \vec{x}_{F_2}(u_2, v_2)$ are arc length scaled along the boundary. The two sub-faces F_1 and F_2 can be situated in a compound face F in 256 different ways and each case is automatically recognized. Fig. 13 gives a particular example. In each case, there is a unique relationship between the global coordinate (u, v) of the geometrical shape function of the compound face and the local coordinates (u_1, v_1) and (u_2, v_2) of the sub-faces. Fig. 13 illustrates this relationship for a particular example. As illustrated, the ratio between length of edges play an important role in establishing the relationship between the coordinate systems. The geometrical shape function $\vec{x}_F(u, v)$ is in general defined as

$$\vec{x}_F(u, v) = \begin{cases} \vec{x}_{F_1}(u_1, v_1) & (u_1, v_1) \in [0, 1]^2 \\ \vec{x}_{F_2}(u_2, v_2) & (u_2, v_2) \in [0, 1]^2 \end{cases} \quad (16)$$

It is easily verified that $\vec{x}_F(u, v)$ is then also arclength scaled along the boundary.

The final result of this recursive definition is that a compound face can be considered as a composite surface consisting of L elementary sub-faces. The compound face is defined on L patches in the unit square. Each patch corresponds with an elementary sub-face. The relation between the global coordinate of the compound face and a local coordinate of an elementary sub-face is defined by the position of the corresponding patch and the orientation of the elementary face in the compound face. An illustration is shown in Fig.16. The local coordinates $(\tilde{u}, \tilde{v}) \in [0, 1]^2$ of an elementary face are related to the global coordinates $\vec{u} = (u, v)^T \in [0, 1]^2$ of the compound face according to the bilinear relation:

$$\vec{u} = \vec{u}_{0,0}(1 - \tilde{u})(1 - \tilde{v}) + \vec{u}_{1,0}\tilde{u}(1 - \tilde{v}) + \vec{u}_{0,1}(1 - \tilde{u})\tilde{v} + \vec{u}_{1,1}\tilde{u}\tilde{v}. \quad (17)$$

Finally, notice that $\vec{x}_F(u, v)$ is continuous and C^1 in the interior of the patches.

The algorithms for the geometrical shape functions of edges and face are implemented in the software of the domain modeler ENDOMO and the grid generator ENGRID. In ENGRID, the geometrical shape function of an elementary face is used to compute a surface grid in that face. The geometrical shape function of a compound face is used in ENDOMO for the projection algorithm of block-faces on the input geometry.

2.4 Topology and Geometry file

The data specifying a blocked flow domain is defined on two files: a topology file and a geometry file. The topological connectivity relations, given by Eqs.(1),(2),(3),(7),(8), are specified on the topology file. The geometry file contains the (x, y, z) position of vertices, and the (x, y, z) position of control points of non-default elementary edges and non-default elementary faces. The topology file and geometry file are the main output files of the domain modeler ENDOMO and the main input files for the grid generator ENGRID.

As an illustration, consider the two partially connected blocks as shown in Fig. 15. The contents of the corresponding topology file is given in Appendix C. The reader may easily check that the contents of this file defines the arrangement of the two blocks as shown in Fig. 15.

3 Domain decomposition

The definition of a blocked flow domain is described in the preceding section. Here we consider some of the main aspects concerning the construction of a blocked flow domain with ENDOMO. The geometry model serves as input to ENDOMO. The geometry model is obtained after surface modeling with ICEM/CFD and consists of pointwise defined surface patches, which abut cleanly, with no gaps, overlaps, doubly defined regions, or non-physical protrusions. However, it is not needed that connected surface patches have a point-to-point match.

In ENDOMO, these pointwise defined surface patches are called discrete surfaces (to emphasize the pointwise definition). In ENDOMO, there is a clear distinction between topological and geometrical elements. The geometrical elements that are used are points, discrete curves, discrete surfaces, nurb curves and nurb surfaces. The topological elements are vertices, edges, faces and blocks. The geometrical elements are used to support the definition of the topological elements. The user constructs geometrical elements which are in turn used to build the topological elements. The main difference between a topological element and a

geometrical element is that a topological element is hierarchically defined by other topological elements. The extraction of points from geometrical elements for the definition of the geometrical shape of a topological element is highly automated in ENDOMO. A geometrical element can also be created from a topological element by using the geometrical shape function of the topological element. A nice example of the simultaneous use of both topological and geometrical elements is the projection algorithm for the creation of faces on the input geometry model. This algorithm is called refacing and described in Section 3.3.

3.1 Geometrical elements

The following set of geometrical elements are available in ENDOMO to support the domain decomposition process of a flow domain in blocks:

- **points.**
- **discrete curves.**
- **discrete surfaces.**
- **nurb curves.**
- **nurb surfaces.**

A discrete curve is defined as a one-dimensional ordered sequence of points. A discrete surface is defined as a two-dimensional ordered sequence of points. The geometry model of a configuration Surface modeling ends with a geometric definition consists of pointwise defined surface patches. These discrete surfaces are the basic elements with which the domain decomposition process starts. Nurb curves and nurb surfaces are mainly used to support the design of other curvilinear edges and faces. A nurb curve (surface) is a non-uniform rational B-spline curve (surface). A nurb curve can be created by specifying its control polygon. Then a cubic B-spline curve is created with a knot sequence based on chord length parametrization (see [18]). The cubic B-spline curve is considered as a nurb curve with constant weights equal to 1. The shape of a nurb curve can be changed by changing the position of a point of the control polygon or by changing a weight at a point of the control polygon. The latter is illustrated in Fig. 20 where a weight at a point of the control polygon has been enlarged. A nurb curve can also be specified by interpolation. Then a cubic B-spline curve is generated that passes through a given set of points (see Fig.20). Thus a nurb curve can be created out of a discrete curve by considering the discrete curve as the defining control polygon or by interpolation. Similarly, a nurb surface can be created out of a discrete surface by considering the discrete surface as the defining control polygon or by interpolation. Fig.21 shows a nurb surface together with its defining control polygon. Fig.22 shows a nurb surface obtained by interpolation of the same control polygon. The shape of a nurb surface can also be changed by changing the position of a point of the control polygon or by changing a weight at a point of the control polygon. An excellent treatise about nurb curves and nurb surfaces can be found in [18],[19].

A discrete curve can be created by interpolation of a nurb curve. A discrete curve can also be created by interpolation of an edge (elementary or compound), using the geometrical shape function of the edge. In both cases, only the number of points has to be specified and a discrete curve will be created consisting of equidistributed points on the edge or nurb curve.

A discrete curve can also be generated by assembling a set of points. A discrete curve can be split into two sub-curves. The splitting occurs at an interior point. Two discrete curves can also be joined together.

A discrete surface can be created by interpolation of a nurb surface. A discrete surface can also be created by interpolation of a face (elementary or compound), using the geometrical shape function of the face. In both cases, the number of points in both coordinate directions have to be specified and an equidistributed point distribution will be created on the nurb surface or face. A discrete surface can also be created by assembling a set of discrete curves (with the restriction that the number of points are the same for all discrete curves). A discrete surface can be split into two sub-surfaces. The splitting occurs at a boundary point. Two discrete surfaces may also be joined together.

In ENDOMO are also available several transformation rules (rotation, translation, scaling, reflection etc.) to transform geometrical elements.

3.2 Construction of topological elements

The extraction of points of discrete curves and discrete surfaces for the definition of the geometrical shape of topological elements is highly automated in ENDOMO.

A vertex can be created by using a point which already exists (such as one which is part of a discrete curve or discrete surface) or by creating a new one in space (either by typing in coordinates or by translation of an existing vertex).

An elementary edge is created by specifying its two vertices. A non-default edge will be automatically created when the position of the two vertices coincide with two different points of a discrete curve. The control points of the non-default edge will become equal to a subset of the points of the discrete curve. A non-default edge will also be automatically created when the position of the two vertices coincide with two points on a row (column) of a discrete surface.

A compound edge is created by specifying its two sub-edges.

An elementary face is created by specifying its four boundary edges. A non-default face will be automatically created when the position of the four corner vertices coincide with four corner points of a rectangular subset of points of a discrete surface. The control points of the non-default face will become equal to this rectangular subset of points. An elementary face can also be created by specifying its four corner vertices. Then the boundary edges of the face, if they do not already exist, are also automatically created.

A compound face is created by specifying its two sub-faces.

A block is created by specifying its six boundary faces. A block can also be created by specifying its eight corner vertices. Then the block-faces and block-edges, if they do not already exist, are automatically created.

3.3 Refacing

An entire aerodynamic configuration is represented in ENDOMO as a collection of pointwise defined surface patches. Block-faces can be easily defined on the surface patches as long as the edges of the block-faces coincide with rows (columns) of points of the surface patches. However, sometimes block-faces are needed on the aerodynamic configuration which cannot be shaped along rows (columns) of the surface patches. Then a block face must be created

by projection on a set of connected surface patches (which may not have a point-to-point match).

Such a projection is performed as follows. The set of surface patches that are needed to project a block-face are joined together to one compound face. Elementary faces are created on each surface patch (the points of the surface patch become the control points of the elementary face). The elementary faces are joined together by compound faces. Thus after all there is only one compound face on which a block-face must be projected. Even when the surface patches do not have a point-to-point match, the geometrical shape function of the compound face still defines a smooth analytical representation on which the projection must be performed, because correction functions automatically provide this smoothness.

Four discrete curves are created in the neighbourhood of the compound face. These discrete curves are initial approximations of the final edges of the projected face. The four discrete curves must define four corner points (which are the initial approximations of the final vertices of the projected face). Furthermore, the number of points on two opposite discrete curves must be the same. These discrete curves are in general created in ENDOMO by interpolation of nurb curves.

Now, ENDOMO provides a facility to project these four discrete curves on the compound face. After projection, the result is a discrete surface with points that are situated on the compound face (and thus on the selected set of surface patches). The points of the discrete surface may then be used as control points of a projected block-face.

This procedure is a nice example of the simultaneous use of topological and geometrical elements in ENDOMO. We will now describe the method to project four discrete curves on a compound face. The method is illustrated in Fig. 18.

Projection Algorithm

Consider a (compound) face F with a geometrical shape function defined by $\vec{x}_F : (u, v) \in [0, 1]^2 \mapsto \mathcal{R}^3$. Call $(u, v) \in [0, 1]^2$ the parameter space \mathcal{P}_{uv} . Consider four discrete curves in the neighbourhood of the compound face, with an equal number of points on opposite curves, as shown in Fig. 18. Then a discrete surface is created on the compound face as follows.

step 1 For each point \vec{x}_c on a discrete curve, a corresponding coordinate point $(u_c, v_c) \in \mathcal{P}_{uv}$ is defined such that

$$(u_c, v_c) = \arg_{(u,v) \in \mathcal{P}_{uv}} \min \| \vec{x}_c - \vec{x}_F(u, v) \|. \quad (18)$$

No Newton-like method can be used to find (u_c, v_c) because $\vec{x}_F(u, v)$ is only a continuous function at the boundary of the patches in \mathcal{P}_{uv} , see Section 2.3. A brute-force two-dimensional variant of the bisection algorithm is used to find (u_c, v_c) . The function $\vec{x}_F(u, v)$ is evaluated on a uniform mesh in parameter space \mathcal{P}_{uv} . Assume that $\vec{x}_{ij} = \vec{x}_F(u_{ij}, v_{ij})$ is the nearest point to \vec{x}_c of the uniform mesh. Then a small rectangular sub-region is defined in \mathcal{P}_{uv} around this point (u_{ij}, v_{ij}) . Again, the geometrical shape function $\vec{x}_F(u, v)$ is evaluated, but now on a uniform mesh in this small rectangular subregion. On this mesh, a new nearest point to \vec{x}_c is determined and the process is repeated until a sufficiently accurate approximation of (u_c, v_c) is found.

step 2 The result of the algorithm described in **step 1** are four discrete curves in parameter space \mathcal{P}_{uv} . These four discrete curves define the four boundary edges of a simply connected sub-region in \mathcal{P}_{uv} . An initial mesh inside this sub-region is generated based on



transfinite interpolation. Also 2D elliptic grid generation, as described in [12], Section 2, can be used to generate an interior mesh.

step 3 The geometrical shape function $\vec{x}_F(u, v)$ is evaluated in the grid points of this initial mesh in this sub-region. The result is a discrete surface with points on the compound face. However, the position on the compound face of the interior points of this discrete surface depend on the parametrization of $\vec{x}_F(u, v)$. The elliptic surface grid generation method, as described in [12], Section 4, can now be directly applied to generate a Laplace mesh on the compound face which is independent of the parametrization. Thus the final result is a discrete surface which only depends on the shape of the surface of the compound face and the position of the computed boundary grid points defined by $\vec{x}_F(u_c, v_c)$.

An illustration of the projection algorithm is shown in Fig.23 and Fig.24. Fig.23 shows four surface patches which are smoothly connected although there is no point-to-point match between the patches. These four patches are joined together in a compound face. Fig.24 shows a projected block-face on the compound face.

A less academic example, shown in Figs. 25,26 concerns the projection of block-faces on the lower part of a wing.

3.4 Interactive domain decomposition

In general, blocking of a flow domain is done in a block-by-block process. The geometrical elements are used to draw the boundary of a block. If a user is satisfied about the geometrical shapes, then a block can be easily generated by specifying eight vertices, and the points of the geometrical elements are automatically used for the geometrical shape of the automatically generated block-edges and block-faces. The control polygon of a nurb curve is often used to draw 3D curves in the interior of the flow domain. For blocks in the interior of the flow domain, it is almost always sufficient to specify the geometrical shape of block-edges only. The geometrical shape of a block-face is then defined as a bilinearly blended Coon's surface bounded by four edge-curves. The points of the surface patches of the geometry model are often used for block-faces on the aerodynamic configuration. Refacing must be applied if this is not possible.

These tasks can only be done within a reasonable time with an advanced graphical user interface (GUI). Some of the most important aspects of the GUI are listed below.

Implementation. The graphical user interface (GUI) of ENDOMO is implemented in C using X Windows and Motif as the interface builder, and GL on the Silicon Graphics workstations as the graphics language. UIMX has been used to develop the GUI. As OpenGL becomes readily available, ENDOMO can be easily ported and then will become available for use on a variety of workstations. The numerical core of ENDOMO has been written in FORTRAN. Only one main FORTRAN subroutine is used for the communication between the GUI and the numerical core. Each domain decomposition task within the GUI, is finally a call to this main subroutine. The result of the task (for example, creating a discrete surface), is stored in a one-dimensional workspace array which the GUI may use to depict the result on the screen. The advantage of this construction is that a **Journal file** can be easily constructed in order to replay an ENDOMO session.



Topology check. It is possible to construct a blocked flow domain which is correct from a topological point of view (i.e. satisfies all topology rules that are used within ENDOMO), but which is unsuitable for multi-block grid generation (i.e. it is not possible to construct a C^0 continuous grid). An example is shown in Fig.17. This figure shows that edge E_3 is a compound edge with sub-edges E_1 and E_2 . Furthermore, edge E_2 and edge E_4 are opposite edges in face F_2 , and edge E_3 and edge E_4 are opposite edges in face F_1 . Thus the grid dimension (i.e. the number of grid cells) must be equal for E_2, E_4 and E_3 so that the grid dimension of edge E_1 must be zero which is not allowed. In ENDOMO it is possible to check if the topology of a blocked flow domain is correct in the sense that it can be used for the construction of a C^0 continuous grid. The algorithm finds out if there are edges with grid dimension zero.

Other checks, for example to ensure that the block decomposition has no gaps or overlaps, are also available in ENDOMO.

Visualization. During domain decomposition, a user does not always want to see all topological and geometrical elements on the computer screen at once. Often, a user wants to see only those elements that are in the vicinity of the block he is creating. ENDOMO provides efficient interactive mechanisms to select the visibility of topological and geometrical elements.

Highlighting visible elements, to visualize the geometrical shape and topological construction of an individual element is also very useful, especially for blocks, compound faces and compound edges.

Boundary conditions All types of boundary conditions, to be used as input for the flow solver ENSOLV, can be specified at elementary faces. The boundary condition data is written on an output file BCDAT. This file can also be an input file, in order to change a specified boundary condition.

4 Grid generation

Grid generation starts with the topology and geometry of a blocked flow domain. A topology and geometry file, created by ENDOMO, are input files for the grid generator ENGRID. Two other files are also important during grid generation: - a grid dimension file which contains the information for the specification of the grid dimensions of the multi-block grid, and - a grid control file which contains the grid control parameters for tuning of the grid. These two files are input and/or output files of a the interactive grid generator.

A batch version of the grid generator is also available. The batch version is operational on a supercomputer (NEC-SX3) and is especially useful to create fine grids.

The general way of working is as follows. Coarse or medium grids are generated during an interactive grid generation session. The generated grids are defined according to the user specified grid dimensions and grid control parameters. When the user is satisfied about the grid quality of the created grids, then the grid dimensions and grid control parameters are written to the grid dimension file and grid control file which are then output files of the interactive grid generator. Next, the grid dimensions are enlarged by a constant factor (the user has to modify, by an editor, only one number on the grid dimension file), and the complete set of four input files (topology file, geometry file, grid dimension file and, grid control file) is sent to the supercomputer where a fine grid is generated by the batch version of the grid generator. This way of working is successful because of the fact that all grid

control parameters have a relative meaning with respect to the grid dimensions. The grid file is the main input file for the flow solver ENSOLV, which also runs on the supercomputer.

4.1 Grid dimension specification

Starting with a blocked flow domain, the first task to be considered during an interactive grid generation session is the specification of the grid dimensions, i.e. the number of grid cells in all blocks, faces and edges. The specification of the grid dimension of a blocked flow domain requires the grid dimension specification of only a few edges. This is due to the constraining effect of the requirement that each grid line in each block must be continuous over any face that the block has in common with any adjacent block (C^0 continuous grid). These constraint relations depend only on the topology: each two opposite edges in a face must have the same grid dimension, and each four opposite edges in a block must have the same grid dimension. This observation makes it possible to subdivide the set of edges $\{E\}$ into disjunct sets (called groups) with the property that the grid dimension of all edges in the same group must be the same, while the grid dimensions of two edges in different groups is generally different. Furthermore, simple sum relations between the grid dimensions of different groups may exist due to the existence of compound edges. If, for instance, a compound edge E with subedges E_1, E_2 belong to the groups G and G_1, G_2 , respectively, then it is clear that the grid dimension of group G is equal to the sum of the grid dimensions of groups G_1 and G_2 .

The groups and the sum relations between the groups are automatically generated from the topology. Suppose that a particular multi-block system contains N groups with M sum relations between the groups. Then there are only $N - M$ independent grid dimensions, and the user has to specify the grid dimensions of only $N - M$ suitable chosen edges in order to define the grid dimensions of all groups, and consequently, of all edges, faces and blocks.

To give an example, consider again the two partially connected blocks in Fig.15. In this case there are six groups, namely

$$\begin{aligned}
 G_1 &= \{E_1, E_4\} \\
 G_2 &= \{E_2, E_5, E_{24}, E_{25}\} \\
 G_3 &= \{E_3, E_6, E_{22}, E_{23}\} \\
 G_4 &= \{E_7, E_8, E_9, E_{10}, E_{11}, E_{12}, E_{13}\} \\
 G_5 &= \{E_{14}, E_{15}, E_{16}, E_{17}\} \\
 G_6 &= \{E_{18}, E_{19}, E_{20}, E_{21}\},
 \end{aligned} \tag{19}$$

and there is one sum relation: $\dim G_1 + \dim G_2 = \dim G_3$. Thus, in this case, the grid dimension of only five suitable chosen edges has to be specified in order to define the grid dimension of all edges, faces and blocks.

4.2 Computational spaces

In a multi-block grid, each topological element has its own computational space. The orientation of the computational space of a topological element is defined by the topological connectivity relations given by Eqs. (1),(2),(3).

For a block B , with $B \mapsto (F_1, F_2, F_3, F_4, F_5, F_6)$, the orientation of the computational space $(\xi_B, \eta_B, \zeta_B) \in [0, 1]^3$ is such that $\xi_B = 0$ at face F_1 , $\xi_B = 1$ at face F_2 , $\eta_B = 0$ at face F_3 , $\eta_B = 1$ at face F_4 , $\zeta_B = 0$ at face F_5 , $\zeta_B = 1$ at face F_6 (see Fig.2).



For a face F , with $F \mapsto (E_1, E_2, E_3, E_4)$, the orientation of the computational space $(\xi_F, \eta_F) \in [0, 1]^2$ is such that $\xi_F = 0$ at edge E_1 , $\xi_F = 1$ at edge E_2 , $\eta_F = 0$ at edge E_3 , $\eta_F = 1$ at edge E_4 (see Fig.3).

Finally, for an edge E , with $E \mapsto (V_1, V_2)$, the orientation of the computational space $\xi_E \in [0, 1]$ is such that $\xi_E = 0$ at vertex V_1 , $\xi_E = 1$ at vertex V_2 .

The size of the computational space of a topological element, i.e. the number of grid cells in each computational coordinate direction, is defined after grid dimension specification. At that moment it is possible to allocate computer memory space to store the grids in the blocks. At the same time, it is also possible to derive automatically the embedding of the computational space of all elementary edges and elementary faces in the computational spaces of the blocks.

For an elementary edge E which is part of block B , the computational space ξ_E of edge E , is related to the computational space (ξ_B, η_B, ζ_B) of block B by a linear transfer relation:

$$\begin{pmatrix} \xi_B \\ \eta_B \\ \zeta_B \end{pmatrix} = \begin{pmatrix} a_0 \\ b_0 \\ c_0 \end{pmatrix} + \xi_E \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix}, \quad (20)$$

where the six constants $a_0 \dots c_1$ are the so-called transfer coefficients.

Similarly, for an elementary face F which is part of block B , the computational space (ξ_F, η_F) of face F , is related to the computational space (ξ_B, η_B, ζ_B) of block B by a linear transfer relation:

$$\begin{pmatrix} \xi_B \\ \eta_B \\ \zeta_B \end{pmatrix} = \begin{pmatrix} a_0 \\ b_0 \\ c_0 \end{pmatrix} + \xi_F \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} + \eta_F \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix}. \quad (21)$$

In this case there are nine transfer coefficients.

As an illustration, Fig.7 shows how the computational space of an elementary edge or elementary face may be embedded in the computational space of a block. The transfer coefficients depend on the topological connectivity relations (which define the orientation of the computational spaces), and the grid dimension of each topological element (i.e. the number of grid cells in each coordinate direction).

The transfer relations defined by Eqs.(20), (21) are extremely important during grid generation: Eq. (20) is used to store the result of a generated grid in an elementary edge, into all the blocks to which this edge belongs to. Similarly, Eq. (21) is used to store the result of a generated grid in an elementary face, into all the blocks of which the face is part of.

The derivation of the transfer relations defined by Eqs.(20),(21) is not trivial. As already mentioned in Section 2.2, the computational space of a topological element is uniquely embedded in the computational space of the topological element to which it belongs. This means that – the computational space of a block-face is uniquely embedded in the computational space of the block to which it belongs, – the computational space of a sub-face is uniquely embedded in the computational space of the compound face to which it belongs, – the computational space of a face-edge is uniquely embedded in the computational space of the face to which it belongs, – the computational space of a sub-edge is uniquely embedded in the computational space of the compound edge to which it belongs. Each embedding defines a basic transfer relation. Eqs.(20),(21) are now found by composing these basic transfer relation in the right order.



4.3 Grid generation in elementary edges

Each elementary edge has a geometrical shape function

$$\vec{x}_E : u \in [0, 1] \mapsto (x, y, z) \in \mathcal{R}^3, \quad (22)$$

where u is the normalized arclength (see Section 2.3). A grid control function u_E of the form:

$$u_E : \xi \in [0, 1] \mapsto u \in [0, 1], \quad (23)$$

maps the computational space onto the parameter space. The orientation of the computational space and parameter space are the same and defined by the topology. A grid distribution function maps the computational space onto the edge-curve and is defined as the composite mapping $\vec{x}_E / o / u_E(\xi) = \vec{x}_E(u_E(\xi))$. Thus the grid points of an edge with N grid cells are computed according to

$$\vec{x}_E / o / u_E(\xi_i), \quad \xi_i = i/N, \quad i = 0 \dots N. \quad (24)$$

The general form of the function $u_E(\xi)$ is taken as

$$u_E(\xi) = \int_0^\xi \exp\left(\sum_{i=0}^4 \alpha_i \eta^i\right) d\eta, \quad (25)$$

where the five coefficients $\alpha_i, i = 0 \dots 4$ are constants. The chosen form of the function u_E implies that the corresponding stretching function, defined as u_E''/u_E' , is a polynomial function. At an elementary edge a user may specify two boundary conditions at each vertex, so that at most four boundary conditions exist. These four boundary conditions, together with the constraint

$$\int_0^1 \exp\left(\sum_{i=0}^4 \alpha_i \eta^i\right) d\eta = 1, \quad (26)$$

are used to compute the five coefficients $\alpha_i, i = 0 \dots 4$. When the number of boundary conditions is less than four, then the five coefficients are still uniquely determined by demanding that the degree of the polynomial stretching function is as low as possible. The grid control function will be the identity, i.e. $u_E(\xi) = \xi$, when no boundary conditions are specified. In that case, the edge will be equidistributed with grid points.

After grid generation in an elementary edge, the computed grid points are automatically stored in all blocks to which the elementary edge belongs.

Grid generation in a compound edge is simply obtained by grid generation in all elementary sub-edges of the compound edge. An algorithm for grid generation in compound edges is therefore not needed.

4.4 Grid generation in elementary faces

The geometrical shape function of an elementary face is also defined in Section 2.3, and given by:

$$\vec{x}_F : (u, v) \in [0, 1]^2 \mapsto (x, y, z) \in \mathcal{R}^3. \quad (27)$$

The orientation of the parameter space $(u, v) \in [0, 1]^2$ is defined by the topology: $u = 0$ at edge E_1 , $u = 1$ at edge E_2 , $v = 0$ at edge E_3 , $v = 1$ at edge E_4 . At the boundary, the



function \vec{x}_F coincides with the geometrical shape functions of the four face-edges (which may be compound edges):

$$\begin{aligned}\vec{x}_F(u, 0) &= \vec{x}_{E_3}(u) \quad , \quad \vec{x}_F(u, 1) = \vec{x}_{E_4}(u), \\ \vec{x}_F(0, v) &= \vec{x}_{E_1}(v) \quad , \quad \vec{x}_F(1, v) = \vec{x}_{E_2}(v).\end{aligned}\tag{28}$$

Thus u and v are normalized arclength along the boundary of the surface (see Section 2.3).

Similarly as for elementary edges, a grid control function maps the computational space onto the parameter space:

$$\vec{u}_F : (\xi, \eta) \in [0, 1]^2 \mapsto (u, v) \in [0, 1]^2.\tag{29}$$

The orientation of the computational space and parameter space are the same. The grid distribution function $\vec{x}_F / o / \vec{u}_F$ maps the computational space onto the surface. The grid points of a face with $N \times M$ grid cells are found by

$$\vec{x}_F / o / \vec{u}_F(\xi_i, \eta_j), \quad \xi_i = i/N, \quad \eta_j = j/M, \quad i = 0 \dots N, \quad j = 0 \dots M.\tag{30}$$

Grid generation in an elementary face is automatically preceded by grid generation in the four face-edges. Thus the grid points along the four face-edges are known and also their corresponding u and v values. What remains is the computation of the grid points in the interior of the face, with the grid points at the four face-edges as Dirichlet boundary conditions.

The computation of the grid control function \vec{u}_F is equivalent with the computation of the two functions

$$u = u(\xi, \eta), \quad v = v(\xi, \eta).\tag{31}$$

These two functions are known at the boundary of the unit square in the computational domain:

$$\begin{aligned}u(\xi, 0) &= u_{E_3}(\xi) \quad , \quad u(0, \eta) = 0, \\ u(\xi, 1) &= u_{E_4}(\xi) \quad , \quad u(1, \eta) = 1, \\ v(0, \eta) &= v_{E_1}(\eta) \quad , \quad v(\xi, 0) = 0, \\ v(1, \eta) &= v_{E_2}(\eta) \quad , \quad v(\xi, 1) = 1.\end{aligned}\tag{32}$$

Note that the functions $u_{E_3}, u_{E_4}, v_{E_1}, v_{E_2}$ are monotonously increasing.

The most simple and robust way to compute (u, v) for values of (ξ, η) in the interior of the unit square is obtained by the same "algebraic straight line transformation" as used in elliptic grid generation, described in [12], Section 2.1. Thus

$$u = u_{E_3}(\xi)(1 - v) + u_{E_4}(\xi)v,\tag{33}$$

$$v = v_{E_1}(\eta)(1 - u) + v_{E_2}(\eta)u.\tag{34}$$

Eq. (33) implies that a grid line $\xi = \text{const.}$ is mapped to the parameter space as a straight line: u is a linear function of v . Eq. (34) implies that a grid line $\eta = \text{const.}$ is also mapped to the parameter space as a straight line: v is a linear function of u . For given values of ξ and η the corresponding u and v values are found as the intersection point of the two straight lines. It can be easily verified that the grid control function which corresponds to this system has a positive Jacobian i.e. $J = u_\xi v_\eta - u_\eta v_\xi > 0$.

The main advantage of this algebraic grid generation method is that it is fast, and robust with respect to grid folding. A disadvantage is that there is no grid control about grid line slopes and grid cell lengths along the boundary. Another disadvantage is that the generated grid is not independent of the parametrization of the face. These disadvantages do not appear in elliptic surface grid generation method as described in [12], Section 4. The algebraic grid generation method in a face is mainly used during the tuning of the grid point distribution along the four face-edges. When a user is satisfied about the boundary grid point distribution, the final grid in the interior of the face is usually computed by the elliptic surface grid generation method.

After grid generation in an elementary face, the computed grid points in the interior of the face are automatically stored in all blocks to which the elementary face belongs.

Grid generation in a compound face is simply obtained by grid generation in all elementary sub-faces of the compound face. An algorithm for grid generation in compound faces is therefore not needed.

4.5 Grid generation in blocks

Grid generation in a block is automatically preceded by grid generation in the six block-faces. Thus, the grid point distribution on the six block faces is known, and what is left to be done is the computation of the grid in the interior of the block with the grid points in the six block-faces as Dirichlet boundary conditions.

The computation of an interior grid in a block is almost always done with the elliptic grid generation method described in [12], Section 5. The method appears to be extremely robust and the quality of the generated grid is good w.r.t. smoothness, grid point distribution and regularity. A disadvantage of the elliptic grid generation method, compared to algebraic grid generation, is that the method is computationally expensive and requires much computer memory usage. Therefore the method is in general not used during interactive grid generation but only in batch during the computation of the final fine grid on the supercomputer.

4.6 Local Grid refinement

A user may want to relax the constraint on grid continuity (C^0 continuous grids) in order to achieve a grid which is efficient of its placement of its grid points. With local grid refinement, the structured grid in a particular block can be refined without changing the grid in the surrounding blocks, so that refined grids can be used in blocks located in regions where large flow gradients are expected.

The grid refinement in a particular block is user-specified by three grid refinement factors in each computational direction of that block.

There is one restriction about the way local grid refinement can be applied. When local grid refinement is applied, there are in general two different grids at a particular internal block-face, which belongs to the two blocks which have this block-face in common. The restriction is that one of the two grids in the block-face is coarse with respect to the other, so that the grid points of the coarse grid is a coarsening in the I and J direction of the grid points of the fine grid. Each grid cell in the coarse grid is then connected to a fixed number of fine grid cells in the fine grid. This property facilitates a flow solver to remain conservative across block-faces: the flux through a coarse grid-cell-face is given by the sum of the fluxes through the corresponding fine grid-cell-faces [9].



An example of an application of local grid refinement is illustrated in Figs. 27, 28, 29, 30.

4.7 Interactive grid generation

The graphical user interface (GUI) of ENGRID is implemented in C using FORMS [20] as the interface builder, and GL on the Silicon Graphics workstations as the graphics language. The numerical core of ENGRID, and also the batch version, has been written in FORTRAN.

During an interactive session, the first action of a user is to read the topology and geometry file of a blocked flow domain.

After that, grid dimensions have to be specified. The mouse is used to select an edge from the screen, and a number is given (by keyboard) which defines the number of grid cells along the edge. With the group concept described in Section 4.1, the program automatically identifies those edges which obtain the same grid dimension value. If there are compound edges, then the sum relations between the groups are used to check if it is possible to compute the dimensions of other edges. If an edge dimension is known then the colour of the edge on the screen is changed and the grid dimension value appears at the middle of the edge.

This process is repeated until the dimensions of all edges are known. The user may then write the grid dimension file which defines the grid dimensions of the multi-block system. If a grid dimension file already exists then the process of specifying the grid dimensions may be skipped and the user can simply read the grid dimension file.

Next, the grid tuning process may start. Along elementary edges, grid control is available only at the two vertices of the edges. The user may specify a grid density ρ at a vertex of an elementary edge by selecting the edge and vertex (via their labels) from the screen by mouse and by defining ρ by keyboard. The result is that the first grid cell length along the edge at the vertex becomes $\rho L/N$ where L is the length of the edge and N is the number of grid cells along the edge. It is also possible to specify a stretching value R at a vertex of an elementary edge. The result is that the ratio between the second and first grid cell length at the vertex becomes $1 + (R/N)$. The user may also "connect" edges. If an edge E_2 at vertex V_2 is connected to an edge E_1 at vertex V_1 then the first grid cell length along edge E_2 at V_2 becomes equal to the first grid cell length along edge E_1 at V_1 . In this way large chains of connected edges may be constructed, and if the grid in the "mother" edge is changed, the grid in all other edges in the chain are then also automatically changed. The program automatically takes care that a chain is not closed. The connection of edges is very useful to construct smooth grids across connected elementary edges. Finally, the user may link an elementary edge to another elementary or compound edge. In that case, the grid point distribution of the elementary edge will become the same as the grid point distribution of the "mother" elementary or compound edge. If the grid point distribution of the mother edge is changed then the grid point distribution of all edges that are linked to that edge are also automatically changed. Linking of edges is especially useful to copy the grid point distribution of a (compound) face-edge to the opposite elementary edge of that face.

During grid tuning, the user selects elementary faces by picking its label with the mouse. ENGRID computes and shows the grid in the selected elementary face. Grid point distribution along edges are always shown via the grid point distribution in an elementary face to which the edge belongs. The grid in the interior of an elementary face is computed with the automatically computed grid points at the four face-edges as Dirichlet boundary conditions. The previously described algebraic and elliptic grid generation methods can be used to compute the interior grid points. With the elliptic method, it is possible to orthogonalize

the interior grid at the boundary of the face. This is done by picking face-edges with the mouse. No controlling parameters are available (and also not needed). Minimal surface grid generation can be applied for default elementary faces (faces with no control points and a face shape defined as a Coon's patch bounded by the four face-edge curves). In that case, the face shape will be no longer a Coon's surface but will become a minimal surface. Minimal surface grid generation [12], Section 3, is especially useful for elementary faces with the four face-edges lying in a plane. Then the minimal surface is a plane surface bounded by the four edges. Finally, a mixed algebraic-elliptic grid generation method is available [13]. This method allows grid tuning by specifying grid-line-slope and/or first-grid-cell height at arbitrarily locations along the four face-edges.

Grid generation in the interior of a block is computed with the automatically computed grid points at the six block-faces as Dirichlet boundary conditions. The previously described elliptic grid generation methods can be used to compute the interior grid points. No controlling parameters are available (and also not needed). However, elliptic grid generation in a block is hardly ever done during interactive grid generation because the method requires much computation time and computer memory.

During the interactive session the user may change at any moment the grid dimension of an edge. The selected grids in the elementary faces shown on the computer screen are then automatically recomputed. The user may also apply local grid refinement at any moment. The refinement of a structured grid in a block is performed by specifying the grid refinement/coarsening factors in the three computational directions of a block.

One interactive session is in general not sufficient to tune the complete grid for a complex configuration. Therefore at the end of a session the user may write a grid dimension file (which contains the specification of the grid dimensions of a multiblock grid) and a grid control file (which contains all grid tuning parameters). Then, in a next session, the user can read these files and continue the grid generation process.

There are of course more additional "tools" in the interactive grid generator. For example, for grid inspection it is necessary to zoom, rotate and translate. These additional tools are evident and need no further description.

5 Applications

The usability of the ENFLOW system is demonstrated for a few rather different types of applications. A hydrodynamic application, created at Delft Hydraulics, is shown in Fig. 31 and Fig. 32. These figures show a multiblock grid in a part of the river Rhine. High grid density is not required at the boundary of the blocks but in the interior of the river. Another, completely different hydrodynamic application is shown in Fig. 33 and Fig. 34.

A multi-block grid about a space capsule is shown in Fig. 35 through Fig. 38. Refacing has been applied at the fore-body of the capsule. The mesh is used for Navier-Stokes calculations.

A multiblock grid of a wing-body-pylon-nacelle configuration is shown in Fig. 39 and Fig. 40. This grid has been used for Euler calculations.

A domain decomposition about a generic fighter is depicted in Fig. 41 and Fig. 42.

Finally, Fig. 43 through Fig. 57 is a rather comprehensive illustration of a multiblock grid for a wing-body-pylon-nacelle-propeller disk configuration. The grid has been used for Navier-Stokes calculations. The total number of grid points is about 2.5 million, the total number of blocks is 106. The challenge of this domain decomposition was to combine a C-type



grid around the wing with a C-type grid around the pylon and with an O-type grid around the nacelle. The C-type grid around the pylon is illustrated in Fig. 46 through Fig. 49. The C-type grid around the wing is illustrated in Fig. 50 and Fig. 51. The O-type grid around the nacelle is shown in Fig. 54, Fig. 55 and Fig. 56. Fig. 57 shows the O-type grid around the wing tip. The interaction of these topologies are illustrated in Fig. 52, Fig. 53 and also in Fig. 54 and Fig. 55. After surface modeling, it took about two weeks man-work to construct this domain decomposition with ENDOMO. Multi-block grid generation with ENGRID took about one week man-work.

Many other multiblock grids for complex geometries have also been constructed at NLR and Fokker, such as transport aircraft configurations, space shuttles, wing bodies, supersonic projectiles, etc.

6 Summary

Block-structured-grid-based methods offer a viable choice for solving viscous flows over complex aerodynamic configurations. Body-fitted structured grids are well suited for resolving the thin viscous layers developing in the vicinity of solid surfaces at high Reynolds numbers typically encountered in flight. A state-of-the-art structured flow solver, like ENSOLV, is very efficient in computing aerodynamic flows in the presence of such embedded boundary layers.

An important step in routine application of structured-grid methodology to CFD applications is the grid generation process. At NLR and Fokker this is done with ENDOMO and ENGRID.

The design of the domain decomposer ENDOMO and grid generator ENGRID has been realized in close relationship between an NLR CFD team as supplier and aerodynamic design and research teams from Fokker and NLR as customer. The partnership between the CFD development team and the design teams has led to a user-friendly GUI based on concepts that automate much of the low level grid generation tasks. The most important concepts that are applied are described in this paper and can be summarized as follows:

- Use of compound edges and compound faces for partial block boundary interfacing.
- The use of topological relations to define degenerations and computational directions.
- Use of Hermite interpolation, for geometrical shape functions of edges and faces, for monotonic interpolation.
- Use of correction functions, for geometrical shape functions of edges and faces, to guarantee C^0 continuous grids.
- Use of Nurb curves (surfaces) to design curvilinear edges (faces) in 3D physical space.
- The automatic extraction of points of geometrical elements during the creation of topological elements.
- The use of an advanced projection algorithm for the projection of block-faces on the original input geometry.
- Automatic topology check during domain decomposition.
- Highly automated grid dimension specification.

- The use of advanced elliptic grid generation methods, for surface grid generation and volume grid generation, with no need to specify non-intuitive controlling parameters.
- The use of local grid refinement to achieve an efficient placement of grid points.
- Both ENDOMO and ENGRID have in common that they provide the user only a limited number of powerful options which are always very intuitive. This makes these systems easy to learn. This is in contrast to CAD/CAM systems which in general contain hundreds of options so that these systems can only be handled by an expert.
- Recomputing a multiblock grid is completely automated when only small changes are made in the input geometry, so that no changes are needed in the multi-block topology. This is achieved via the **Journal** file of ENDOMO and the input files of the batch version of ENGRID.

A Piecewise cubic Hermite interpolation for curves

Consider a set of control points $\{\vec{x}_i = (x, y, z)_i^T \mid i = 0 \dots N\}$. We wish to construct a smooth C^1 curve $\vec{x} : u \in [0, 1] \mapsto \mathcal{R}^3$ which is passing through the set of control points with a geometrical shape as one would intuitively expect. Furthermore, spurious oscillations should be prevented. For this reason, cubic spline interpolation is not safe. Instead, piecewise cubic Hermite interpolation is applied. The extra freedom to model the tangent vectors is used to prevent unwanted oscillations. The parameter u is defined as normalized arc length.

Compute the distance between succeeding control points:

$$\bar{d}_i = \|\vec{x}_i - \vec{x}_{i-1}\|, \quad i = 1 \dots N. \quad (35)$$

Define the length of the curve by

$$L = \sum_{i=1}^N \bar{d}_i, \quad (36)$$

and the normalized distances as

$$d_i = \bar{d}_i / L, \quad i = 1 \dots N. \quad (37)$$

Define the knot sequence $\{u_i \mid i = 0 \dots N\}$ by $u_0 = 0$ and

$$u_i = u_{i-1} + d_i, \quad i = 1 \dots N. \quad (38)$$

Hence, $0 = u_0 < u_1 < \dots < u_N = 1$. Patch i is defined as the interval $[u_{i-1}, u_i]$. In patch i , we relate a local variable $\alpha \in [0, 1]$ to the global variable u by

$$u = u_{i-1} + \alpha(u_i - u_{i-1}) = u_{i-1} + \alpha d_i. \quad (39)$$

For the moment, suppose that the tangent vectors $\{\vec{x}_{u_i} = \frac{d\vec{x}}{du}(u_i), i = 0 \dots N\}$ are known. How these tangent vectors are modeled is shown below. The shape of the curve at patch i is then defined as

$$\vec{x}(\alpha) = \vec{x}_{i-1}H_0(\alpha) + \vec{x}_iH_1(\alpha) + d_i\vec{x}_{u_{i-1}}H_2(\alpha) + d_i\vec{x}_{u_i}H_3(\alpha), \quad (40)$$



where H_0, H_1, H_2, H_3 are cubic Hermite interpolation functions defined as

$$\begin{aligned} H_0(\alpha) &= (1 + 2\alpha)(1 - \alpha)^2, \\ H_1(\alpha) &= (3 - 2\alpha)\alpha^2, \\ H_2(\alpha) &= \alpha(1 - \alpha)^2, \\ H_3(\alpha) &= (\alpha - 1)\alpha^2, \end{aligned} \quad (41)$$

with $0 \leq \alpha \leq 1$.

It can be easily verified that $\frac{d\bar{x}}{du}(u_i-) = \frac{d\bar{x}}{du}(u_i+) = \bar{x}_{u_i}$, so that the piecewise cubic curve $\bar{x}(u)$ is indeed C^1 .

The tangent vectors $\{\bar{x}_{u_i}, i = 0 \dots N\}$ are computed as follows. Define the chord vectors

$$\bar{x}_{u_{i-\frac{1}{2}}} = \frac{\bar{x}_i - \bar{x}_{i-1}}{d_i}, \quad i = 1 \dots N. \quad (42)$$

Note that $\|\bar{x}_{u_{i-\frac{1}{2}}}\| = L$. The tangent vectors at the interior knots $i = 1 \dots N - 1$ are modeled as

$$\bar{x}_{u_i} = \bar{x}_{u_{i-\frac{1}{2}}} c_i + \bar{x}_{u_{i+\frac{1}{2}}} (1 - c_i), \quad i = 1 \dots N - 1, \quad (43)$$

with

$$c_i = \frac{\|\bar{x}_i - \bar{x}_{i-1}\|^2}{\|\bar{x}_i - \bar{x}_{i-1}\|^2 + \|\bar{x}_{i+1} - \bar{x}_i\|^2} = \frac{d_i^2}{d_i^2 + d_{i+1}^2}, \quad i = 1 \dots N - 1. \quad (44)$$

If $\|\bar{x}_i - \bar{x}_{i-1}\| \ll \|\bar{x}_{i+1} - \bar{x}_i\|$ then $c_i \approx 0$ and $\bar{x}_{u_i} \approx \bar{x}_{u_{i+\frac{1}{2}}}$. This implies that high curvature will be restricted to small patches which is a behaviour as one would intuitively expect. Spurious oscillations are also prevented.

Quadratic end conditions are used to compute the end tangent vectors \bar{x}_{u_0} and \bar{x}_{u_N} . The quadratic end conditions require that the cubic polynomial function $\bar{x}(\alpha)$ is a quadratic function of α in patch 1 and in patch N . It is easily verified that this implies that

$$\bar{x}_{u_0} = 2\bar{x}_{u_{\frac{1}{2}}} - \bar{x}_{u_1}, \quad \bar{x}_{u_N} = 2\bar{x}_{u_{N-\frac{1}{2}}} - \bar{x}_{u_{N-1}}. \quad (45)$$

Fig.19 illustrates that cubic Hermite interpolation prevents spurious oscillations, in contrast to cubic spline interpolation.

B Piecewise bicubic Hermite interpolation for surfaces

Consider a set of control points $\{\bar{x}_{i,j} = (x, y, z)_{i,j}^T \mid i = 0 \dots N, j = 0 \dots M\}$. We wish to construct a smooth C^1 surface $\bar{x} : (u, v) \in [0, 1]^2 \mapsto \mathcal{R}^3$ which is passing through the control points with a geometrical shape as one would intuitively expect. As for curves, spurious oscillations should be prevented. For this reason, bicubic spline interpolation is not safe. Instead, piecewise bicubic Hermite interpolation is applied. The extra freedom to model the tangent vectors and twist vectors is used to prevent unwanted oscillations.

Consider a row of points $\{\bar{x}_{i,j} \mid i = 0 \dots N\}$ with $j \in \{0 \dots M\}$ fixed. This row of points is considered as a discrete curve and it is therefore possible to compute a knot sequence $\{u_{i,j} \mid i = 0 \dots N\}$ in exactly the same way as described in the previous section. In the same way, consider a column of points $\{\bar{x}_{i,j} \mid j = 0 \dots M\}$ with $i \in \{0 \dots N\}$ fixed, and compute the knot sequence $\{v_{i,j} \mid j = 0 \dots M\}$.

To construct a smooth surface, one knot sequence is used for all rows and for all columns. These two knot sequences are obtained by averaging:

$$u_i = \frac{1}{M+1} \sum_{j=0}^M u_{i,j}, \quad i = 0 \dots N, \quad v_j = \frac{1}{N+1} \sum_{i=0}^N v_{i,j}, \quad j = 0 \dots M. \quad (46)$$

Patch (i, j) is defined as the rectangle $[u_{i-1}, u_i] \times [v_{j-1}, v_j]$. In patch (i, j) we relate local variables $(\alpha, \beta) \in [0, 1]^2$ to the global variables (u, v) by

$$u = u_{i-1} + \alpha d_i^u, \quad v = v_{j-1} + \beta d_j^v, \quad (47)$$

with $d_i^u = u_i - u_{i-1}$ and $d_j^v = v_j - v_{j-1}$.

For the moment, suppose that the tangent vectors $\vec{x}_{u,i,j} = \frac{\partial \vec{x}}{\partial u}(u_i, v_j)$, $\vec{x}_{v,i,j} = \frac{\partial \vec{x}}{\partial v}(u_i, v_j)$, and twist vectors $\vec{x}_{uv,i,j} = \frac{\partial^2 \vec{x}}{\partial u \partial v}(u_i, v_j)$ are known for all knots (i, j) . How these tangent and twist vectors are modeled is shown below.

The shape of the surface at patch (i, j) is then defined as

$$\vec{x}(\alpha, \beta) = (H_0(\alpha), H_1(\alpha), H_2(\alpha), H_3(\alpha)) M_{i,j}^H \begin{pmatrix} H_0(\beta) \\ H_1(\beta) \\ H_2(\beta) \\ H_3(\beta) \end{pmatrix}, \quad (48)$$

where the matrix $M_{i,j}^H$ is defined by

$$M_{i,j}^H = \begin{pmatrix} \vec{x}_{i-1,j-1} & \vec{x}_{i-1,j} & d_j^v \vec{x}_{v_{i-1,j-1}} & d_j^v \vec{x}_{v_{i-1,j}} \\ \vec{x}_{i,j-1} & \vec{x}_{i,j} & d_j^v \vec{x}_{v_{i,j-1}} & d_j^v \vec{x}_{v_{i,j}} \\ d_i^u \vec{x}_{u_{i-1,j-1}} & d_i^u \vec{x}_{u_{i-1,j}} & d_i^u d_j^v \vec{x}_{uv_{i-1,j-1}} & d_i^u d_j^v \vec{x}_{uv_{i-1,j}} \\ d_i^u \vec{x}_{u_{i,j-1}} & d_i^u \vec{x}_{u_{i,j}} & d_i^u d_j^v \vec{x}_{uv_{i,j-1}} & d_i^u d_j^v \vec{x}_{uv_{i,j}} \end{pmatrix}. \quad (49)$$

From these definitions, it can be easily verified that the piecewise bicubic surface $\vec{x}(u, v)$ is C^1 .

The tangent vectors $\vec{x}_{u,i,j}$ are computed as follows (the tangent vectors $\vec{x}_{v,i,j}$ are computed in the same way). Define the chord vectors

$$\vec{x}_{u_{i-\frac{1}{2}},j} = \frac{\vec{x}_{i,j} - \vec{x}_{i-1,j}}{d_i^u}, \quad i = 1 \dots N, \quad j = 0 \dots M, \quad (50)$$

and use the same non-linear averaging formula as used for curves, thus

$$\vec{x}_{u,i,j} = \vec{x}_{u_{i-\frac{1}{2}},j} c_{i,j} + \vec{x}_{u_{i+\frac{1}{2}},j} (1 - c_{i,j}), \quad i = 1 \dots N - 1, \quad j = 0 \dots M. \quad (51)$$

with

$$c_{i,j} = \frac{\|\vec{x}_{i,j} - \vec{x}_{i-1,j}\|^2}{\|\vec{x}_{i,j} - \vec{x}_{i-1,j}\|^2 + \|\vec{x}_{i+1,j} - \vec{x}_{i,j}\|^2}, \quad i = 1 \dots N - 1, \quad j = 0 \dots M. \quad (52)$$

Quadratic end conditions are used again to compute the end tangent vectors.

A modification of Adini's method [19] is used to compute the twist vectors. Consider patch (i, j) with local variables (α, β) . Assume that the tangent vectors \vec{x}_u, \vec{x}_v are known at the four corner points of the patch. Introduce the abbreviate notation $\vec{x}_{00} = \vec{x}_{i-1,j-1}$, $\vec{x}_{10} = \vec{x}_{i,j-1}$, $\vec{x}_{01} = \vec{x}_{i-1,j}$, $\vec{x}_{11} = \vec{x}_{i,j}$. Use Eqs.(48),(49) to find $\vec{x}_\alpha(0,0) = d_i^u \vec{x}_{u_{i-1,j-1}}$, $\vec{x}_\alpha(1,0) = d_i^u \vec{x}_{u_{i,j-1}}$,



$\bar{x}_\alpha(0, 1) = d_i^u \bar{x}_{u_{i-1,j}}$, $\bar{x}_\alpha(1, 1) = d_i^u \bar{x}_{u_{i,j}}$, $\bar{x}_\beta(0, 0) = d_j^v \bar{x}_{v_{i-1,j-1}}$, $\bar{x}_\beta(1, 0) = d_j^v \bar{x}_{v_{i,j-1}}$, $\bar{x}_\beta(0, 1) = d_j^v \bar{x}_{v_{i-1,j}}$, $\bar{x}_\beta(1, 1) = d_j^v \bar{x}_{v_{i,j}}$. Compute the boundary curves of the patch by cubic Hermite interpolation. Thus, for example

$$\bar{x}(\alpha, 0) = \bar{x}_{00}H_0(\alpha) + \bar{x}_{10}H_1(\alpha) + \bar{x}_\alpha(0, 0)H_2(\alpha) + \bar{x}_\alpha(1, 0)H_3(\alpha), \quad (53)$$

and the other three boundary curves are computed by similar formulas. Define the shape of the surface patch as a bilinearly blended Coon's patch

$$\bar{x}(\alpha, \beta) = (1 - \alpha, \alpha) \begin{pmatrix} \bar{x}(0, \beta) \\ \bar{x}(1, \beta) \end{pmatrix} + (1 - \beta, \beta) \begin{pmatrix} \bar{x}(\alpha, 0) \\ \bar{x}(\alpha, 1) \end{pmatrix} - (1 - \alpha, \alpha) \begin{pmatrix} \bar{x}_{00} & \bar{x}_{01} \\ \bar{x}_{10} & \bar{x}_{11} \end{pmatrix} \begin{pmatrix} 1 - \beta \\ \beta \end{pmatrix}. \quad (54)$$

Compute the corner twist vectors $\bar{x}_{\alpha\beta}(0, 0)$, $\bar{x}_{\alpha\beta}(1, 0)$, $\bar{x}_{\alpha\beta}(0, 1)$, $\bar{x}_{\alpha\beta}(1, 1)$ from Eq.(54). Use Eq.(47) to find the corresponding twist vectors w.r.t. the global variables (u, v) : $\bar{x}_{uv_{i-1,j-1}} = \bar{x}_{\alpha\beta}(0, 0)/d_i^u d_j^v$ etc..

Thus at the four corners of each patch, an estimation is found for the twist vector \bar{x}_{uv} . Consider an interior knot (i, j) . Then four estimations for the twist vector $\bar{x}_{uv_{i,j}}$ are found in respectively patches (i, j) , $(i + 1, j)$, $(i, j + 1)$ and $(i + 1, j + 1)$. Write those estimations as respectively \bar{x}_{uv}^{SW} , \bar{x}_{uv}^{SE} , \bar{x}_{uv}^{NW} , \bar{x}_{uv}^{NE} . A similar averaging procedure as applied for tangent vectors is used to define a unique value $\bar{x}_{uv_{i,j}}$:

$$\bar{x}_{uv_{i,j}} = \frac{\bar{x}_{uv}^{SW} A_{i,j}^2 + \bar{x}_{uv}^{SE} A_{i+1,j}^2 + \bar{x}_{uv}^{NW} A_{i,j+1}^2 + \bar{x}_{uv}^{NE} A_{i+1,j+1}^2}{A_{i,j}^2 + A_{i+1,j}^2 + A_{i,j+1}^2 + A_{i+1,j+1}^2}, \quad (55)$$

where $A_{i,j}$ is the area of patch (i, j) defined as

$$A_{i,j} = 0.5 \| (\bar{x}_{i,j} - \bar{x}_{i-1,j-1}) \wedge (\bar{x}_{i-1,j} - \bar{x}_{i,j-1}) \|. \quad (56)$$

This non-linear averaging procedure guarantees that large changes in twist will be restricted to small patches. At a boundary knot, there are only two estimations for the twist vector. It is evident how the averaging procedure must be applied in that case. At the four corner knots, only one estimation is available and averaging is thus not needed.

C Example of a topology file

The contents of the following topology file corresponds with the two-blocks as shown in Fig. 15.

```
' TOPOLOGY FILE OF TWO PARTIALLY CONNECTED BLOCKS '
' ENDOMO '
' 2.10 '
' 95-07-21 '
' 12:22:52 '
NUMBER OF BLOCKS:
2
BLOCK <-----FACES-----> IDENT
1 5 3 9 8 13 12 'BLOCK'
2 7 6 1 4 11 10 'BLOCK'
```



NUMBER OF ELEMENTARY FACES:

12

FACE	<-----EDGES----->				IDENT
1	2	5	7	8	'FACE'
2	1	4	8	9	'FACE'
4	24	25	10	11	'FACE'
5	22	23	12	13	'FACE'
6	14	15	2	24	'FACE'
7	5	25	16	17	'FACE'
8	22	3	18	19	'FACE'
9	23	6	20	21	'FACE'
10	15	17	8	11	'FACE'
11	7	10	14	16	'FACE'
12	13	9	19	21	'FACE'
13	12	7	18	20	'FACE'

NUMBER OF COMPOUND FACES:

1

FACE	<--FACES-->	<-----EDGES----->				IDENT
3	1	2	7	9	3	6 'FACE'

NUMBER OF ELEMENTARY EDGES:

23

EDGE	<VERTICES>		IDENT
1	1	2	'EDGE'
2	1	3	'EDGE'
4	4	5	'EDGE'
5	4	6	'EDGE'
7	3	6	'EDGE'
8	1	4	'EDGE'
9	2	5	'EDGE'
10	7	9	'EDGE'
11	8	10	'EDGE'
12	11	13	'EDGE'
13	12	14	'EDGE'
14	3	7	'EDGE'
15	1	8	'EDGE'
16	6	9	'EDGE'
17	4	10	'EDGE'
18	11	3	'EDGE'
19	12	2	'EDGE'
20	13	6	'EDGE'
21	14	5	'EDGE'
22	11	12	'EDGE'
23	13	14	'EDGE'
24	7	8	'EDGE'
25	9	10	'EDGE'

NUMBER OF COMPOUND EDGES:

2

EDGE	<--EDGES-->	<VERTICES>	IDENT
3	1 2	2	3 'EDGE'
6	4 5	5	6 'EDGE'

References

- [1] N.P. Weatherill, P.R. Eiseman, J. Hauser, J.F. Thompson (eds), Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, Proceedings of the 4th International Conference, Swansea, Wales, Pineridge Press, 1994.
- [2] A.S. Arcill, J. Hauser, P.R. Eiseman, J.F. Thompson (eds), Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, Proceedings of the 3th International Conference, Barcelona, Spain, Elsevier Science Publishers, 1991.
- [3] Y. K. Choo (ed), Surface Modeling, Grid Generation, and Related Issues in Computational Fluid Dynamic (CFD) Solutions, NASA CP 3291. Proceedings of a workshop held at NASA Lewis, Cleveland, Ohio, 1995.
- [4] R.E. Smith (ed), Software Systems for Surface Modeling and Grid Generation, NASA CP 3143, Proceedings of a workshop held at NASA Langley, Hampton, VA, 1992.
- [5] N.P. Weatherill (ed), Grid Generation, VKI Lecture Series 1994-02, 1994.
- [6] N.P. Weatherill (ed), Numerical Grid Generation, VKI Lecture Series 1990-06, 1990.
- [7] J.W. Boerstoel, ENFLOW, A System of CFD Codes for Industrial CFD Analysis of Flows Around Aircraft Including Propulsion Systems Modelling, NLR-CR-93519, 1993.
- [8] J.W. Boerstoel, S.P. Spekreijse, P.L. Vitagliano, The Design of a System of Codes for Industrial Calculations of Flows around Aircraft and other Complex Aerodynamic Configurations, NLR-TP-92190, AIAA paper 92-2619, 10th Applied Aerodynamics Conference, Palo Alto, 1992.
- [9] A. Kassies, R. Tognaccini, Boundary Conditions for Euler Equations at Internal Block Faces of Multiblock Domains, using Local Grid Refinement, NLR-TP-90134, AIAA paper 90-1590, 1990.
- [10] J.F. Dannenhoffer, Multiblock Grid Generation, in VKI-Lecture-Series 1994-02: Grid Generation, 1994.
- [11] V.N. Vatsa, M.D. Sanetrik, E.B. Parlette, Block-Structured Grids for Complex Aerodynamic Configurations: Current Status, in Proceedings Surface Modeling, Grid Generation, and Related Issues in Computational Fluid Dynamic (CFD) Solutions, NASA-CP-3291, 1995.
- [12] S.P. Spekreijse, J.W. Boerstoel, Multiblock Grid Generation. Part 1: Elliptic Grid Generation Methods for Structured Grids. This VKI-Lecture-Serie, 1996.



- [13] S.P. Spekreijse, J.W. Boerstoel, J.L. Kuyvenhoven, M.J. van der Marel, Surface Grid Generation for Multi-block Structured Grids, NLR-TP-92267, in: Proceedings of the First European Computational Fluid Dynamics Conference 1992, Ch. Hirsch et al (eds), vol 2, pp. 937-944, Elsevier Science Publishers, 1992.
- [14] S.P. Spekreijse, J.W. Boerstoel, New Concepts for Multi-Block Grid Generation for Flow Domains around Complex Aerodynamic Configurations, NLR-TP-91046, in: Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, Proceedings of the 3th International Conference, Barcelona, Spain, Elsevier Science Publishers, 1991.
- [15] R. Hagmeijer, Grid Adaptation based on Modified Anisotropic Diffusion Equations formulated in the Parametric Domain, Journal of Computational Physics, 115, 169-183, 1994.
- [16] R. Hagmeijer, Adaptive generation of structured grids. This VKI-Lecture-Serie, 1996.
- [17] M.J. van der Marel, Evaluation of ENGRID and ENDOMO. Delft Hydraulics report Q997, 1991.
- [18] G. Farin, Curves and Surfaces for Computer Aided Geometric Design, A practical guide, Academic Press, San Diego, 1990.
- [19] F. Yamaguchi, Curves and Surfaces in Computer Aided Geometric Design, Springer Verlag, Berlin, 1988.
- [20] M.H. Overmars, FORMS, A Graphical User Interface Toolkit for Silicon Graphics Workstations, Department of Computer Science, Utrecht University, Utrecht, Netherlands, version 2.3, 1995.

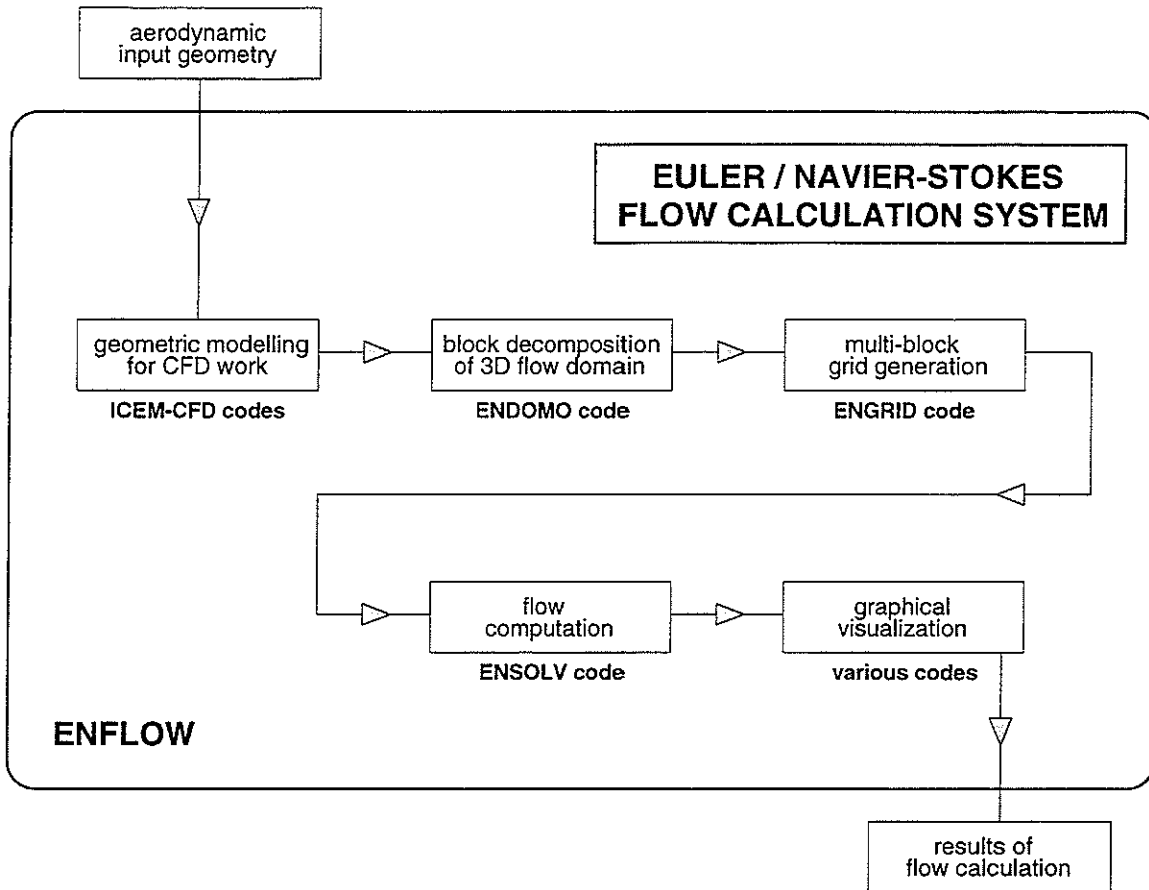


Figure 1: The NLR ENFLOW system.

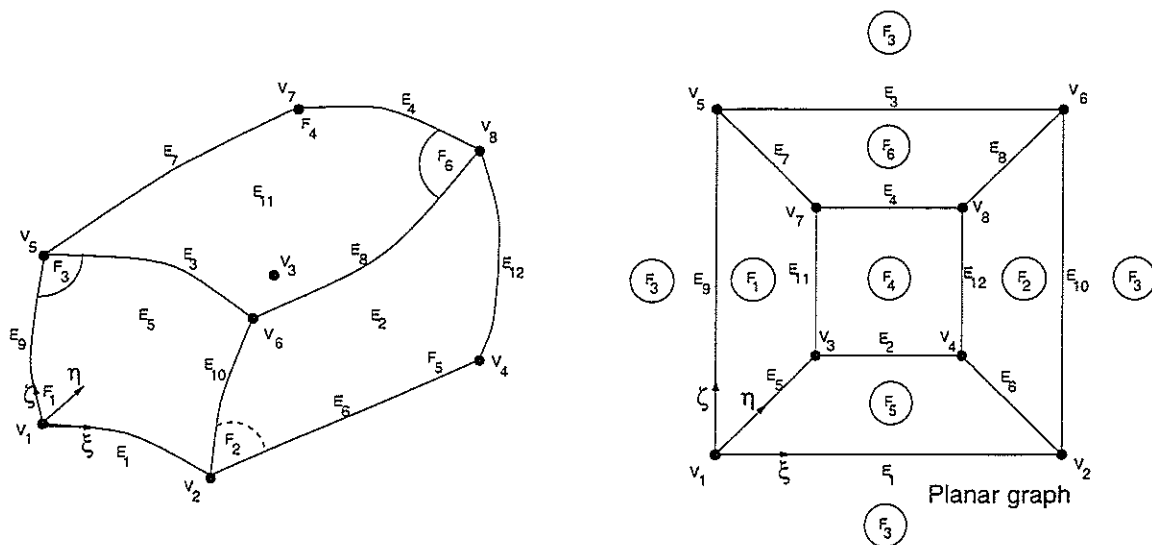


Figure 2: Topology of a block.

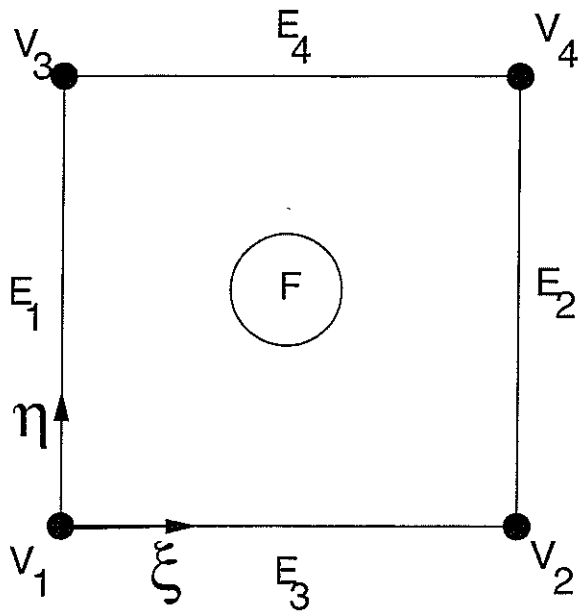


Figure 3: Topology of a face.

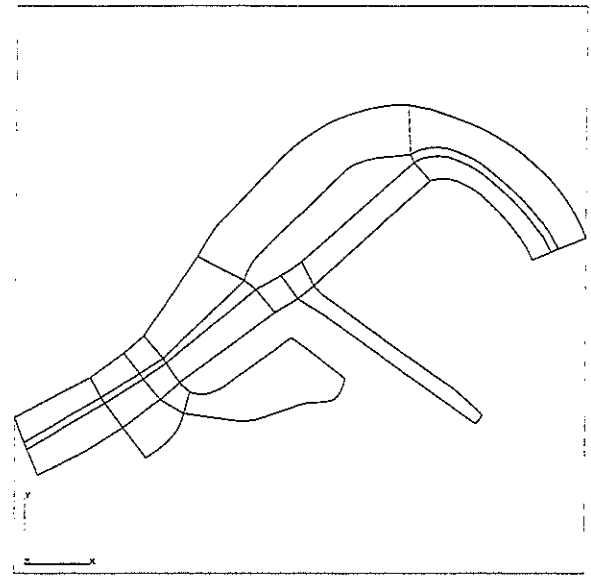


Figure 4: Complete block boundary interfacing.

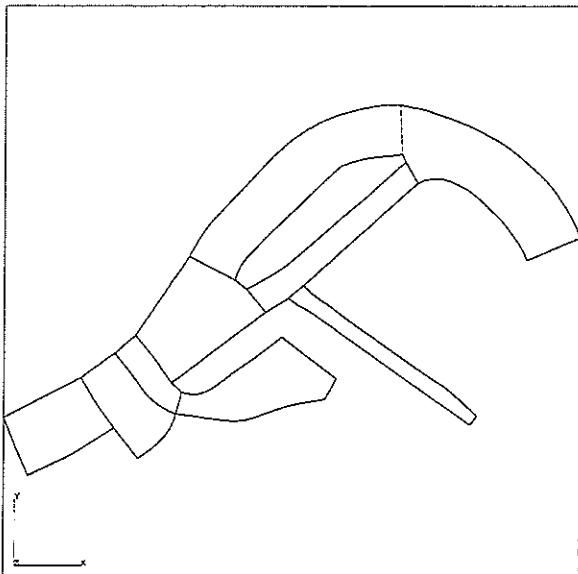


Figure 5: Partial block boundary interfacing.

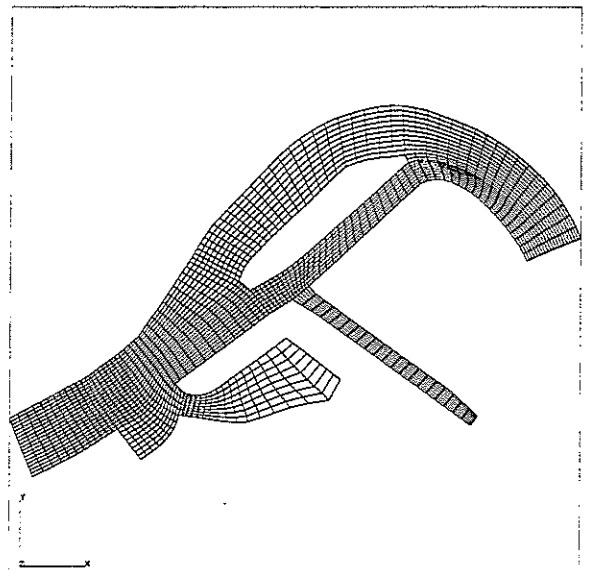


Figure 6: Block structured grid.

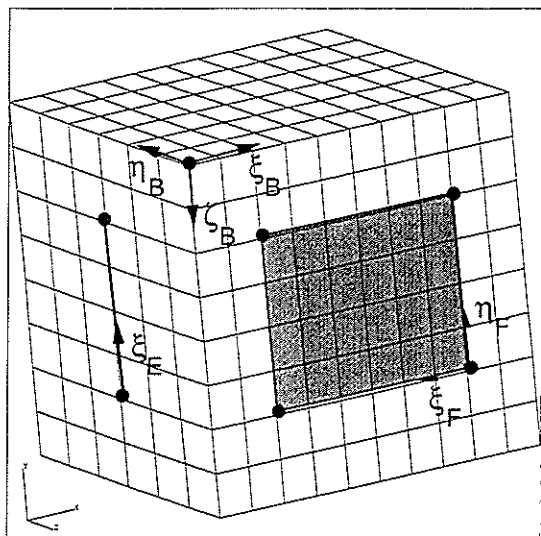


Figure 7: Embedding of an edge and face into the computational space of a block.

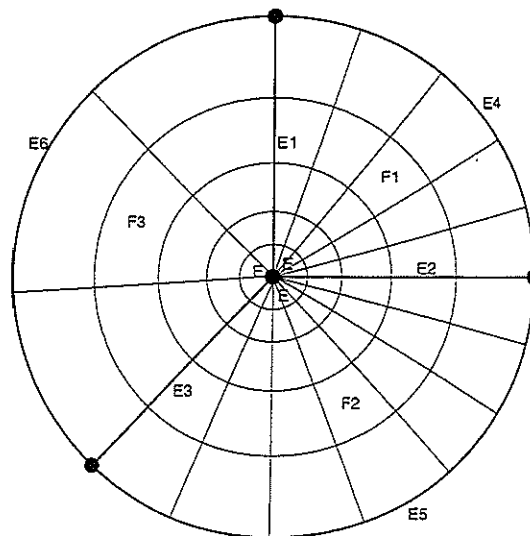


Figure 8: Number of grid points along collapsed edge E is not unique.

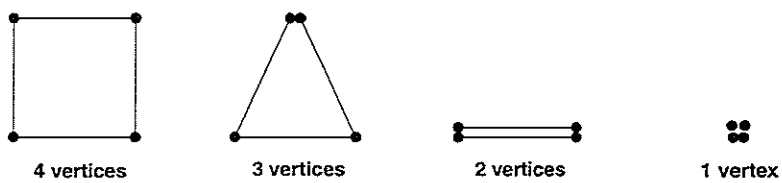


Figure 9: Allowed types of elementary faces.

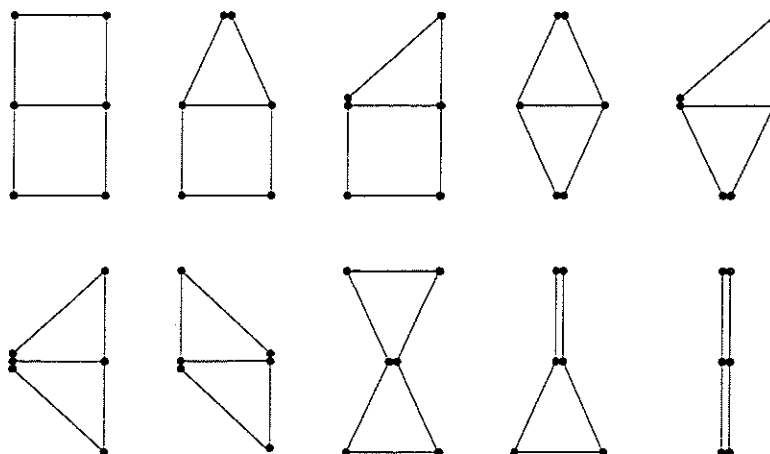


Figure 10: Allowed types of compound faces.

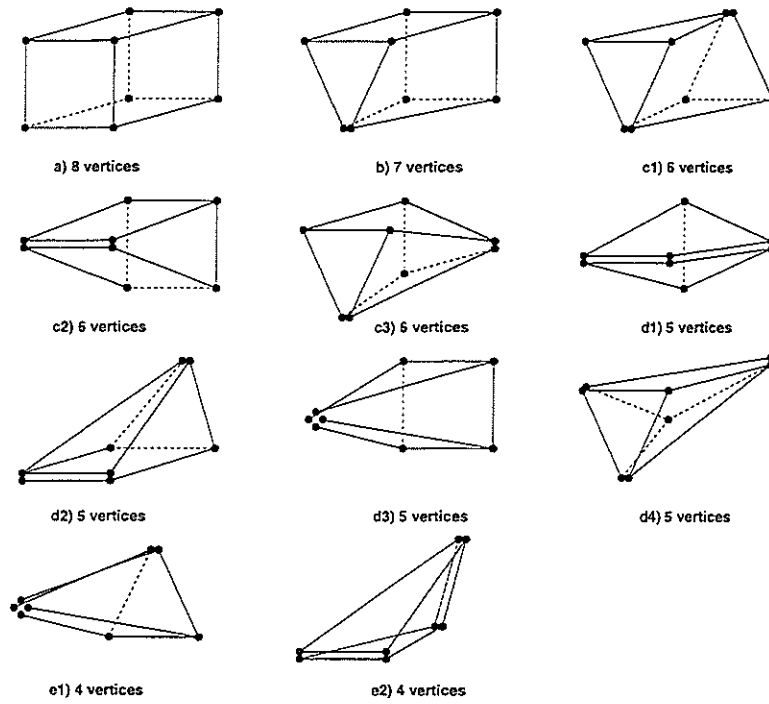


Figure 11: Allowed types of blocks.

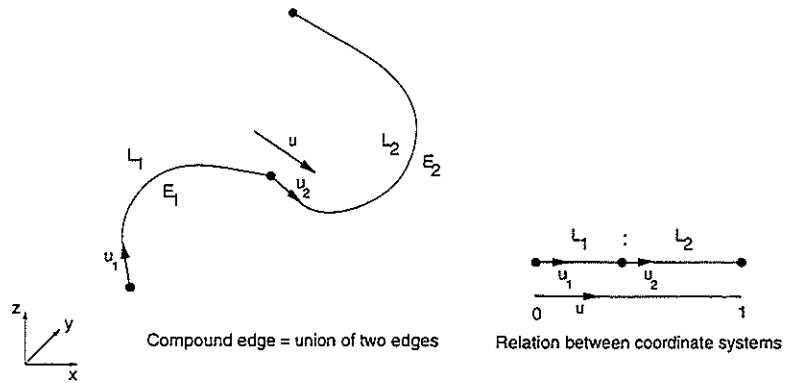


Figure 12: Relation between the coordinate systems of two sub-edges in a compound edge.

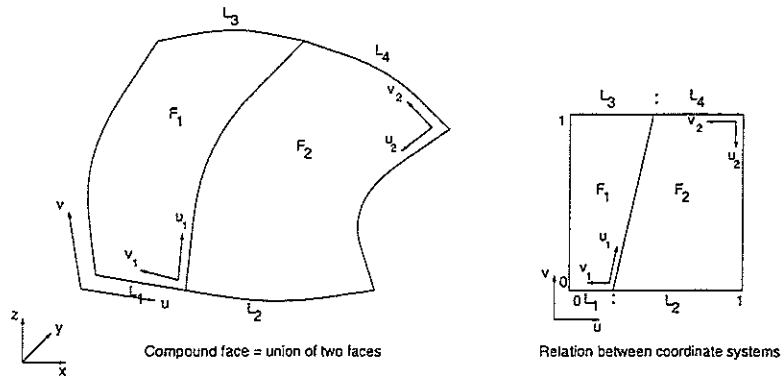


Figure 13: Example of the relation between the coordinate systems of two sub-faces in a compound face.

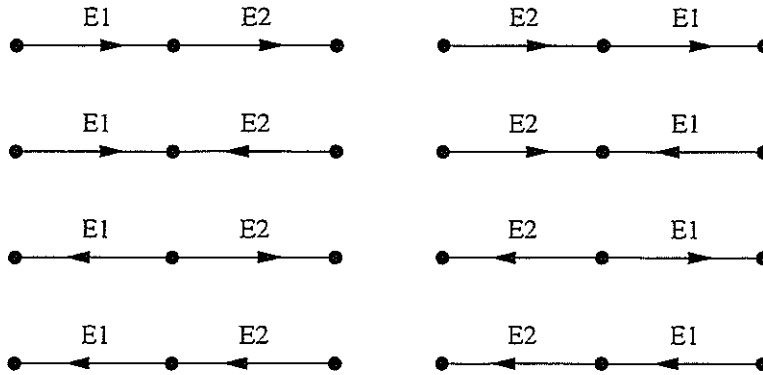
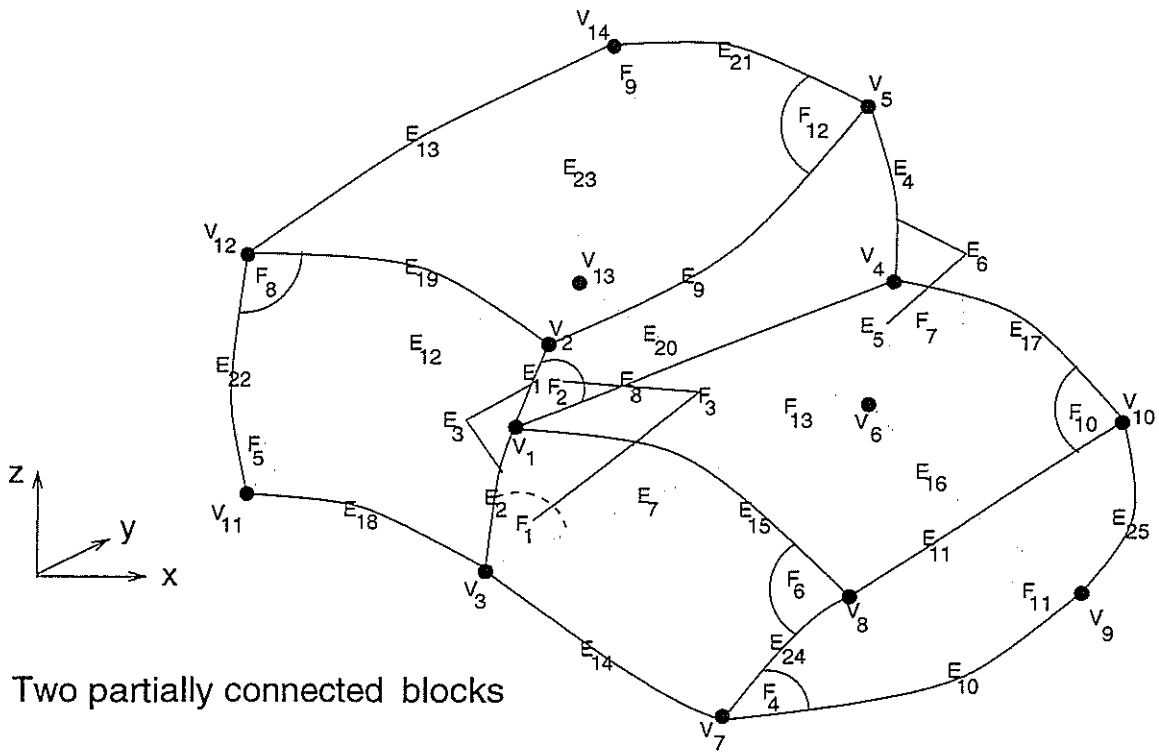


Figure 14: The eight possibilities how two sub-edges E_1 and E_2 can be situated in a compound edge.



Two partially connected blocks

Figure 15: Topology of two partially connected blocks.

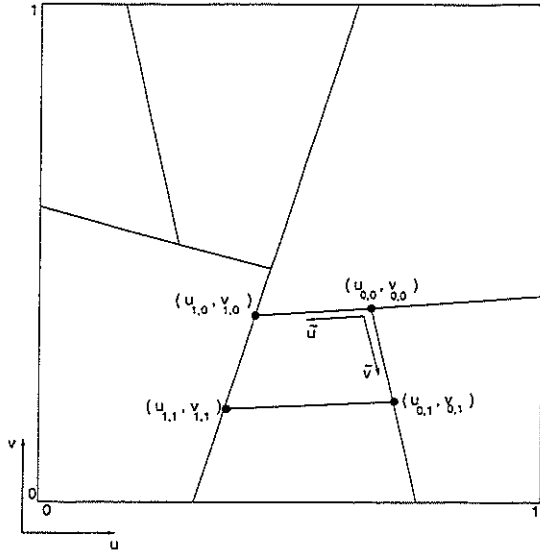


Figure 16: Relation between the coordinate systems of elementary faces in a compound face. The compound face consists of 7 elementary faces.

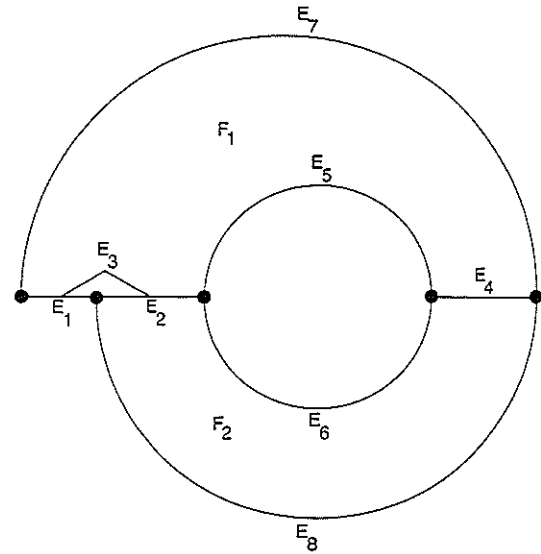


Figure 17: Example of a topological construction which can not be used for multi-block grid generation.

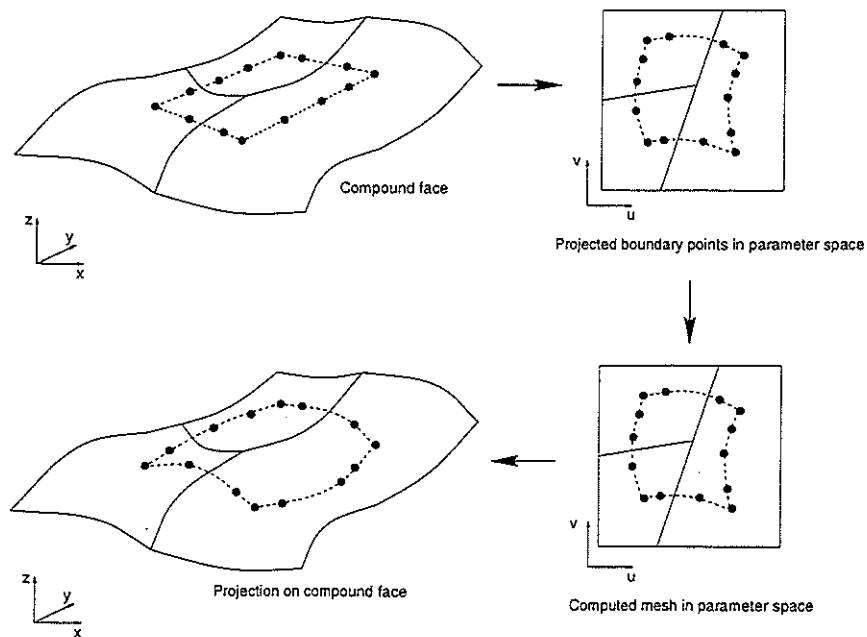


Figure 18: Illustration of the projection algorithm on a compound face.

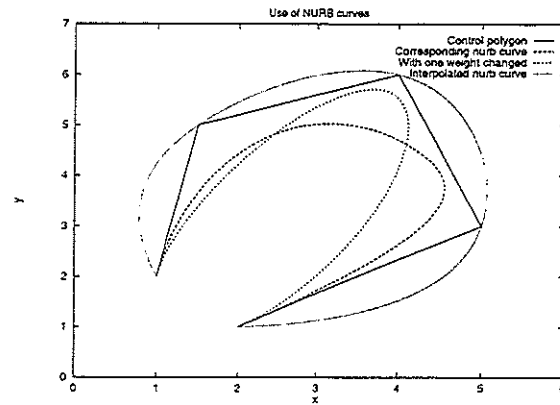
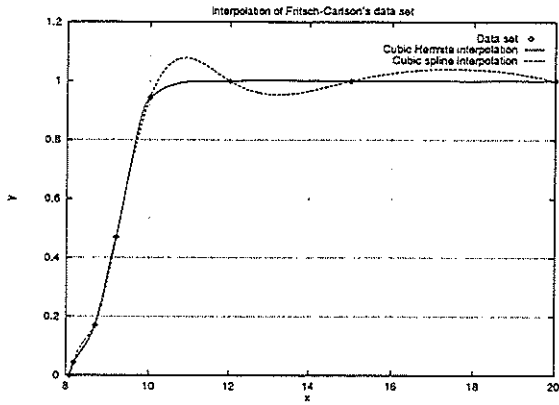


Figure 19: Comparison of cubic Hermite and cubic spline interpolation.

Figure 20: Non-Uniform Rational B-spline curve and its control polygon.

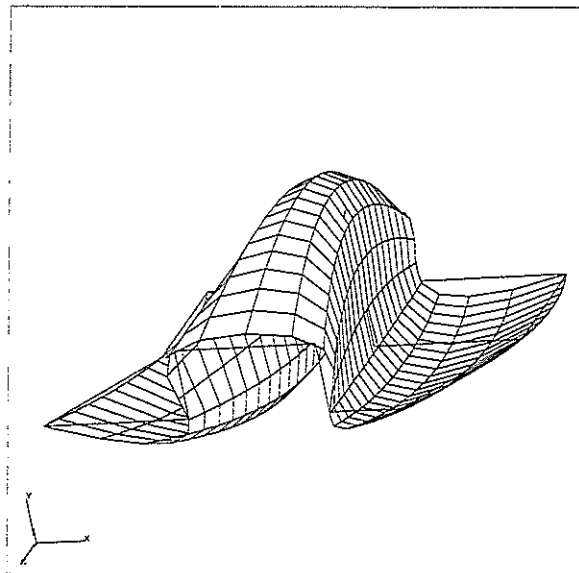
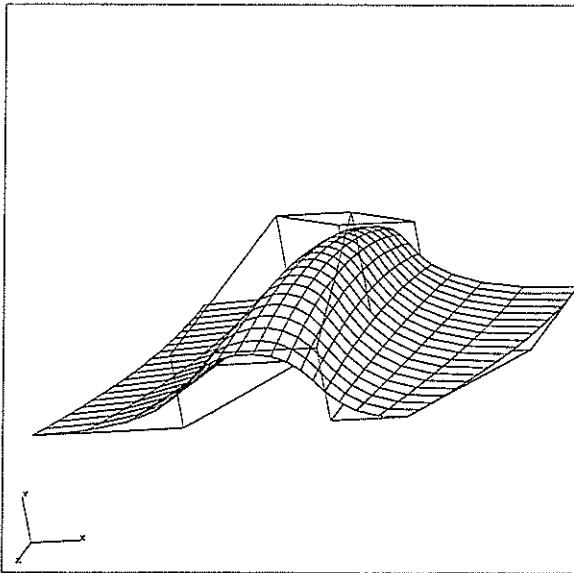


Figure 21: Non-Uniform Rational B-spline surface and its control polygon.

Figure 22: NURB surface obtained by interpolation.

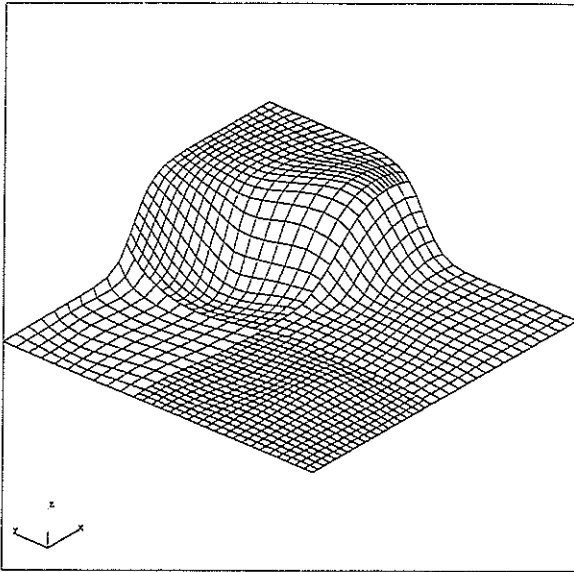


Figure 23: Four surface patches joined together in a compound face.

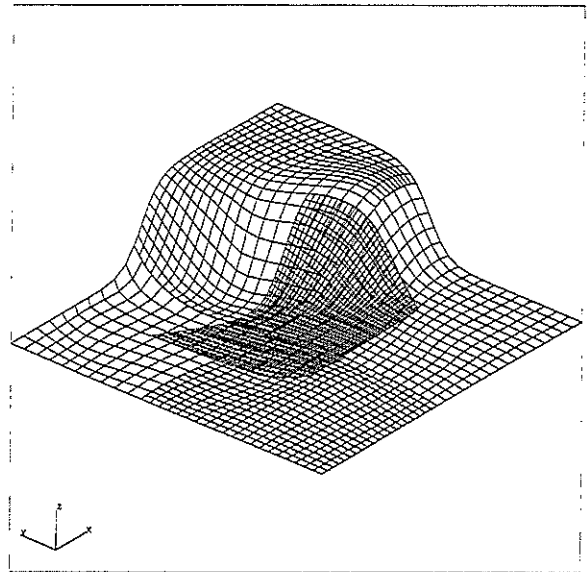


Figure 24: Projected block-face on the compound face.

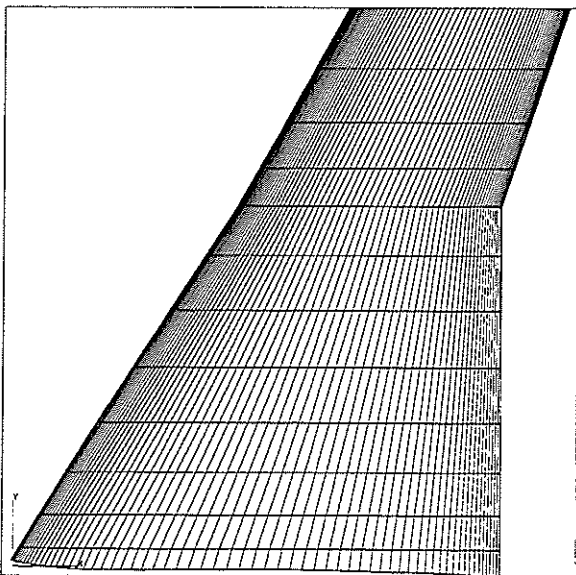


Figure 25: Lower part of a wing.

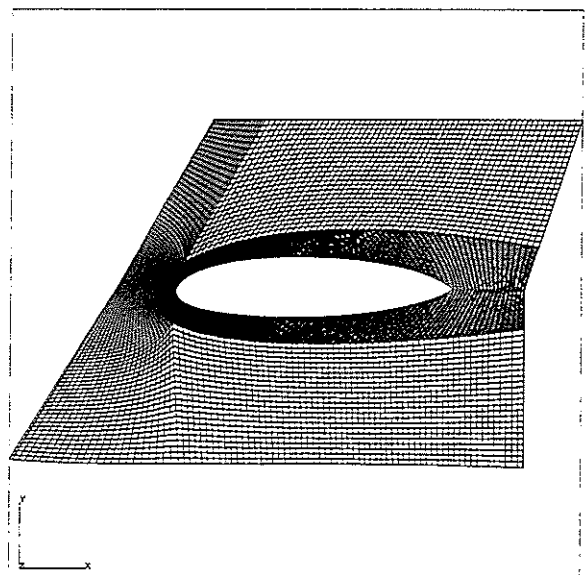


Figure 26: Projected block-faces on wing.

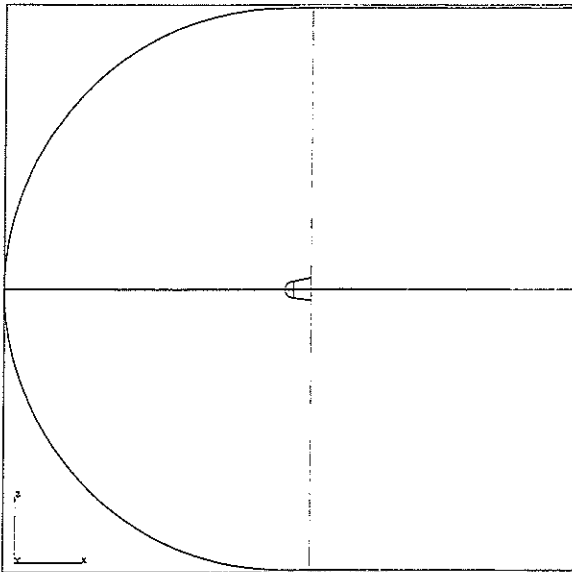


Figure 27: Block decomposition around RAE2822 airfoil.

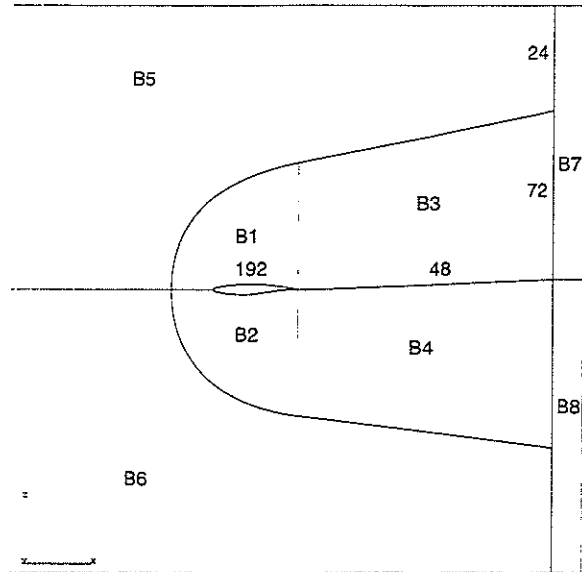


Figure 28: Block decomposition and grid dimension for RAE2822 airfoil.

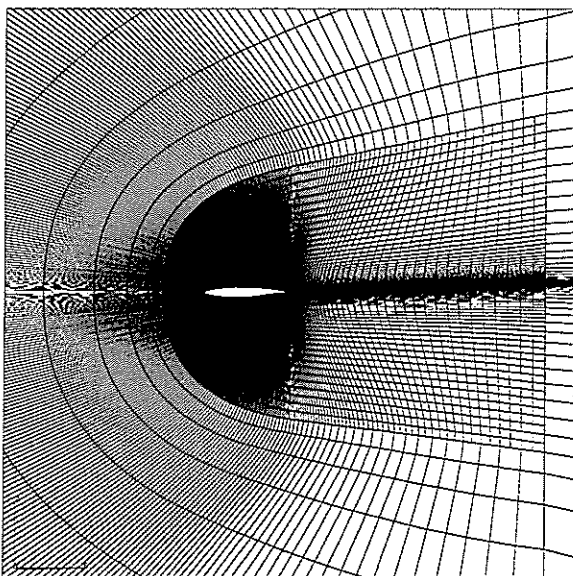


Figure 29: Local grid refinement.

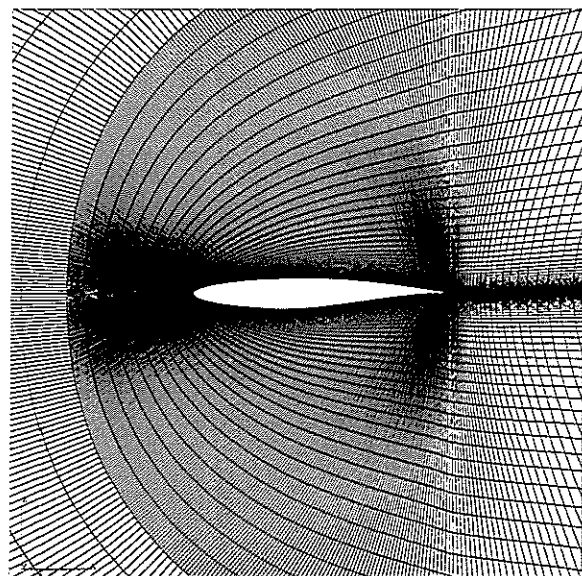


Figure 30: Grid near RAE2822 airfoil.

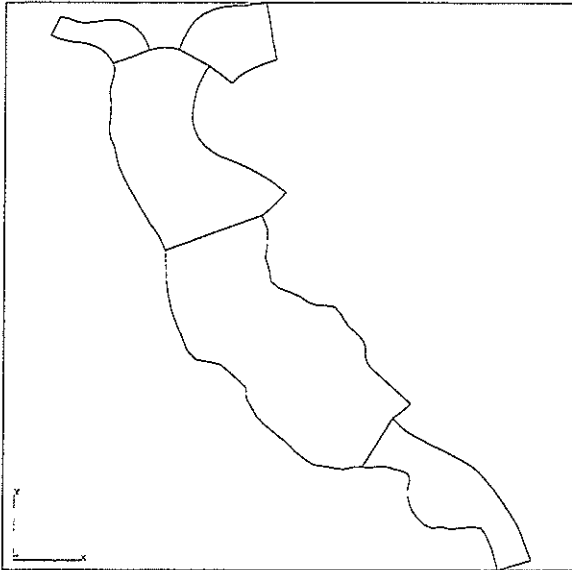


Figure 31: Block decomposition in part of river Rhine.

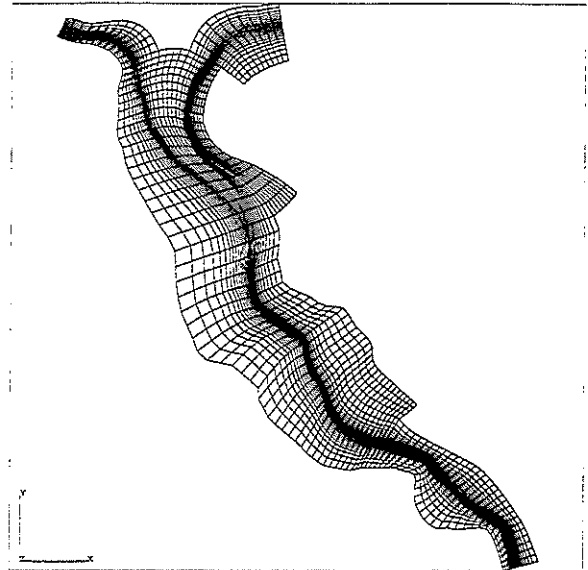


Figure 32: Corresponding grid.

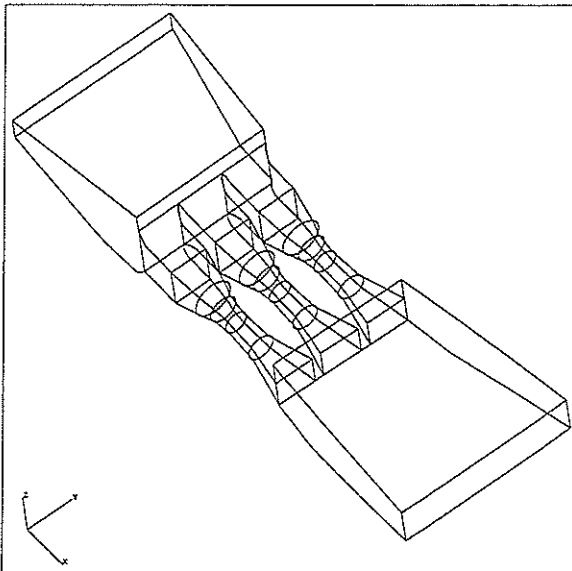


Figure 33: Block decomposition of a water-power station.

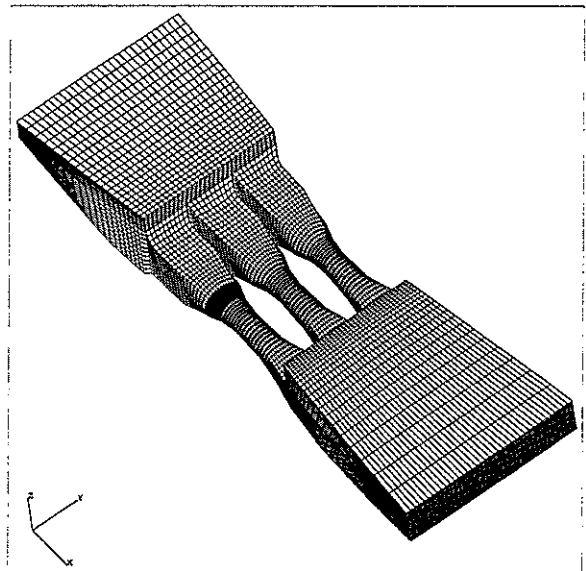


Figure 34: Corresponding grid.

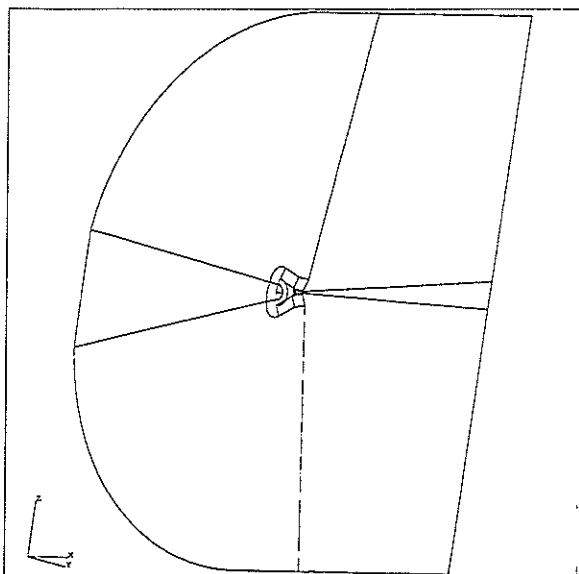


Figure 35: Illustration of a block decomposition of a space capsule.

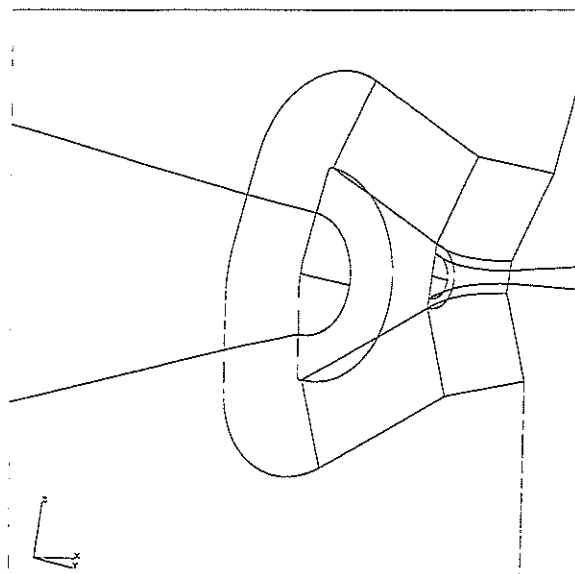


Figure 36: Close-up of block decomposition.

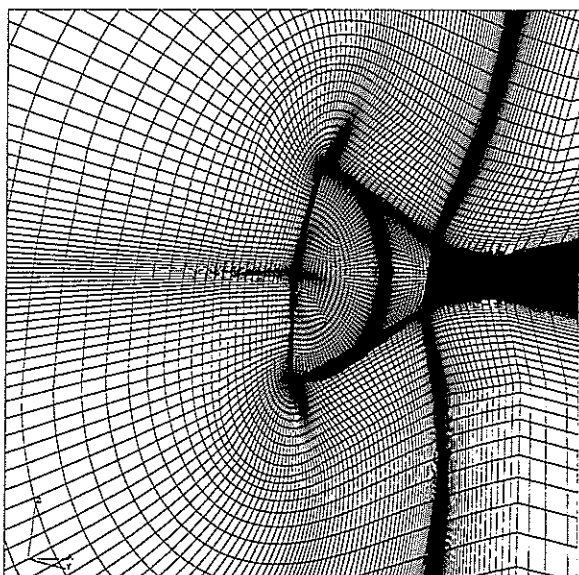


Figure 37: Grid.

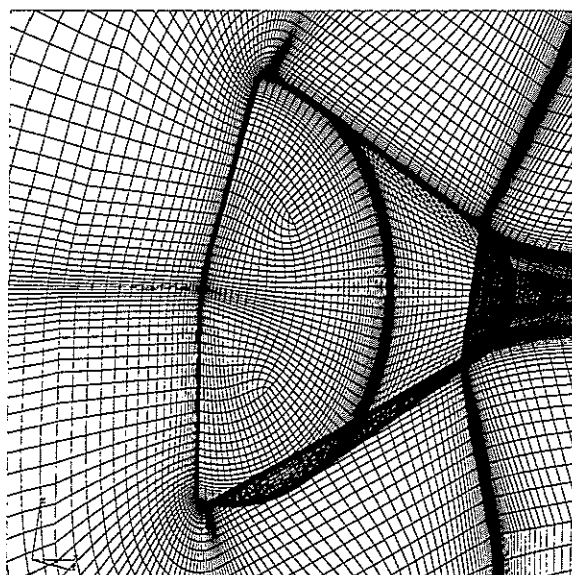


Figure 38: Close-up of grid.

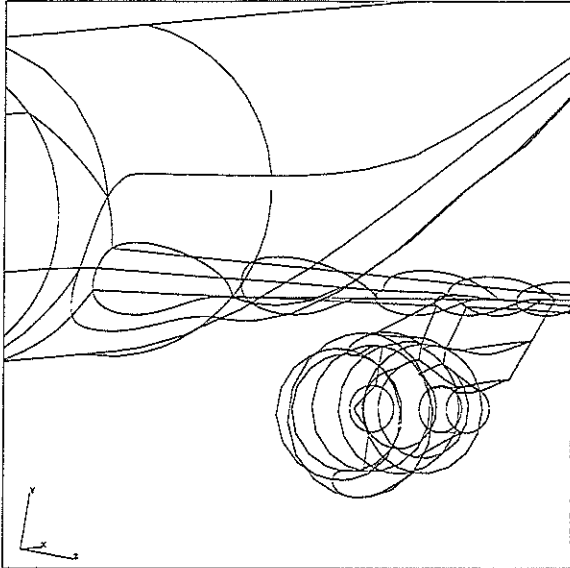


Figure 39: Block decomposition on the surface of a wing body nacelle pylon configuration.

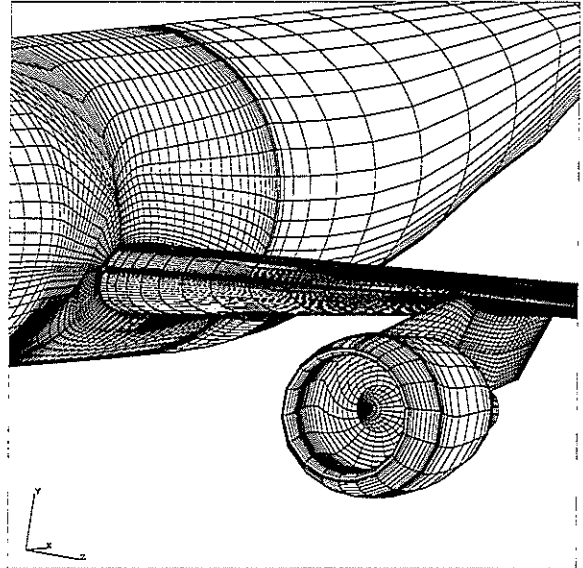


Figure 40: Corresponding surface grid.

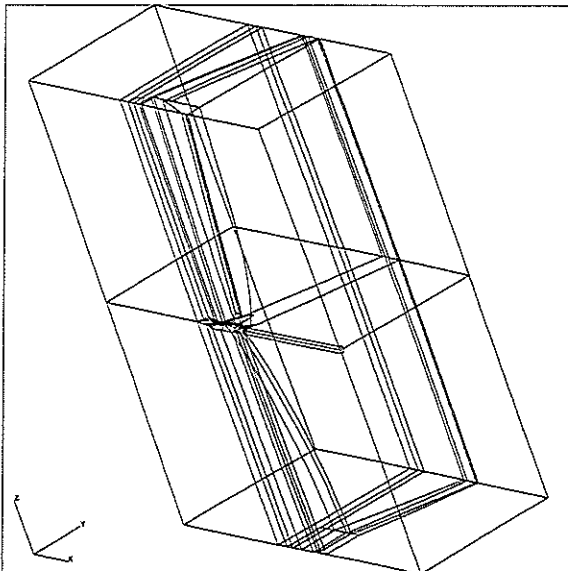


Figure 41: Block decomposition about a generic fighter.

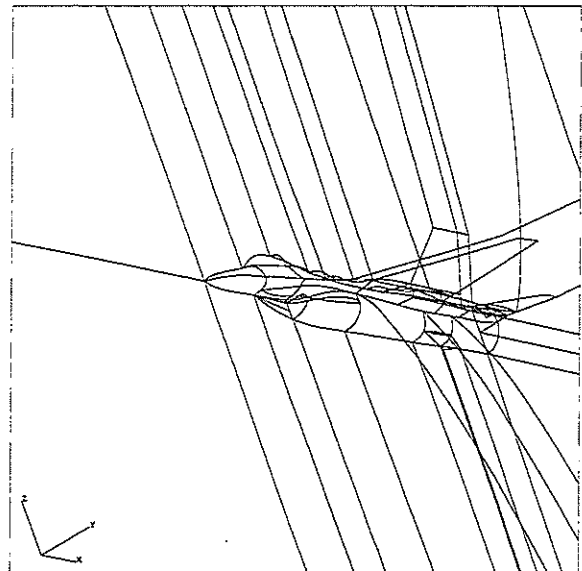


Figure 42: Close-up.

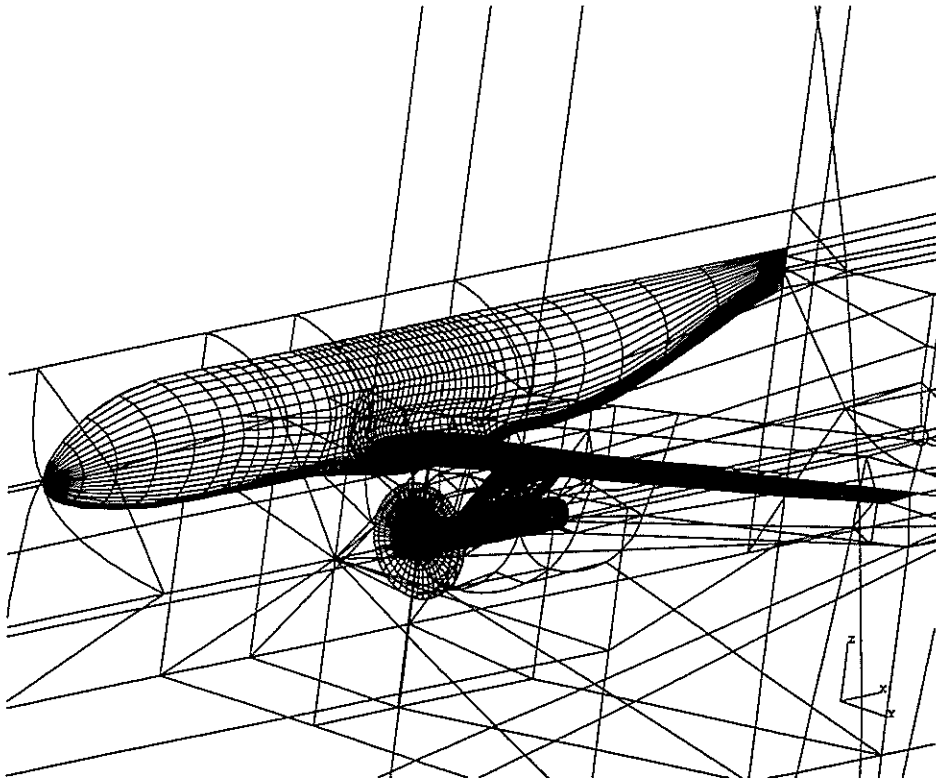


Figure 43: Block decomposition and medium surface grid of a configuration consisting of a fuselage, wing, pylon, nacelle, and propeller disk.

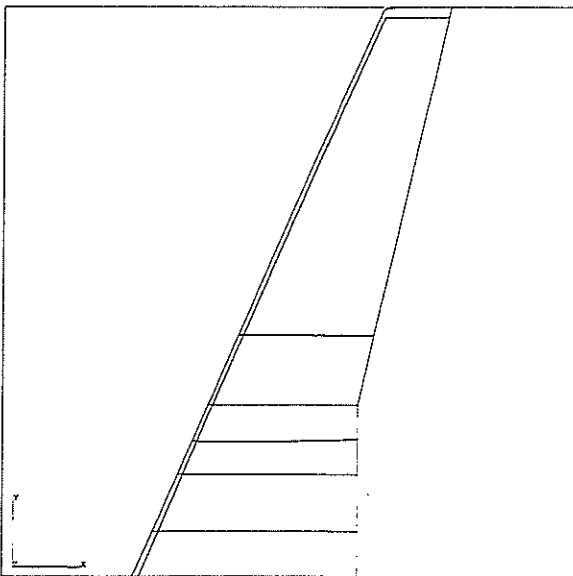


Figure 44: Block decomposition on the wing upper surface.

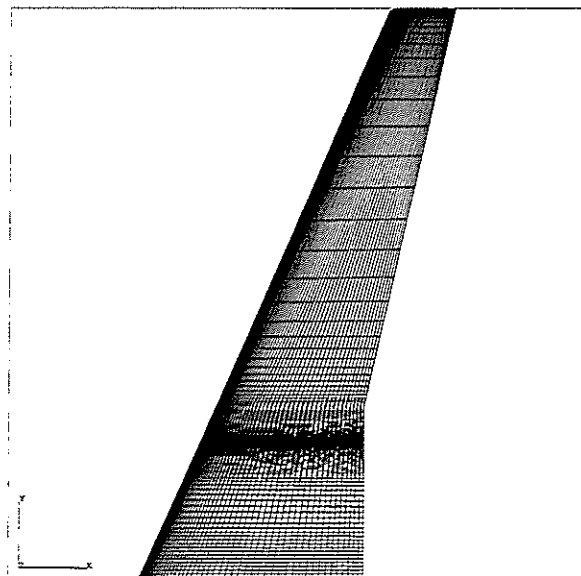


Figure 45: Fine grid on the wing upper surface.

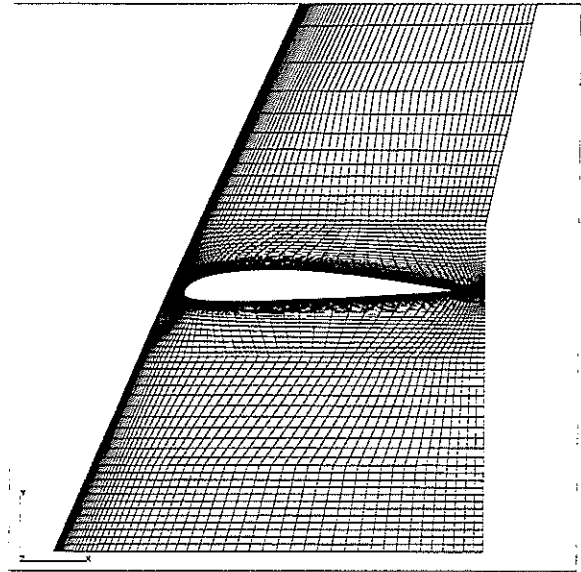
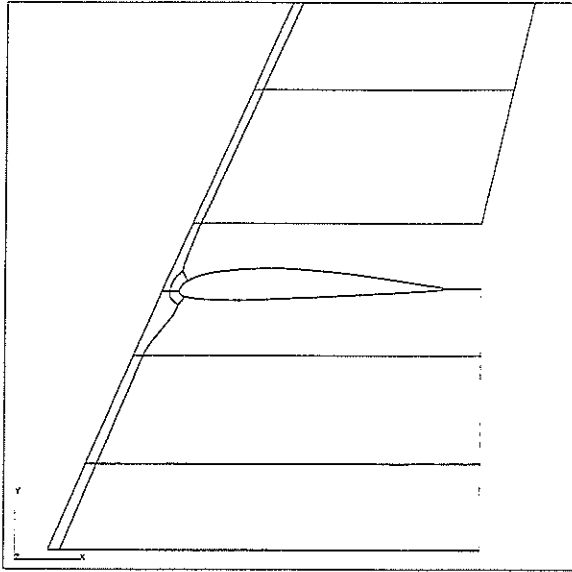


Figure 46: Block decomposition on the wing lower surface.

Figure 47: Fine grid on the wing lower surface.

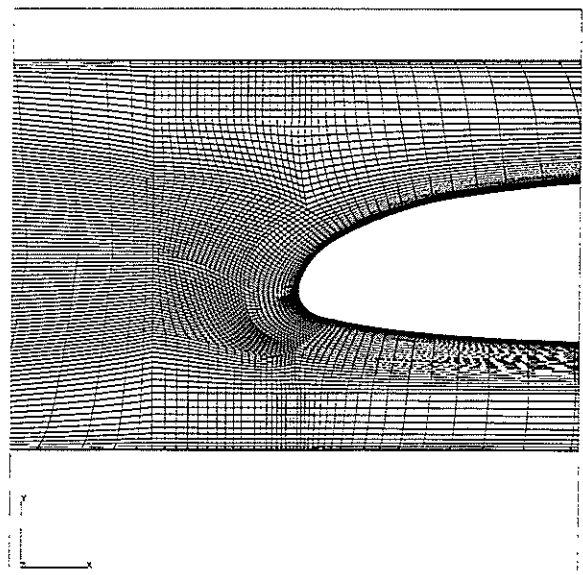
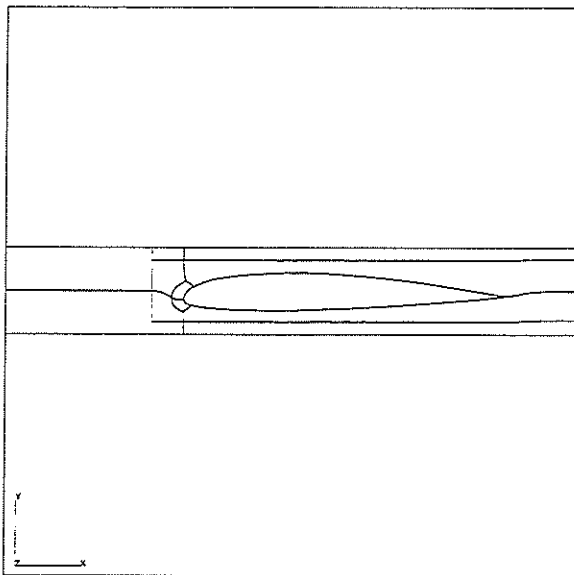


Figure 48: Block decomposition on the nacelle upper surface.

Figure 49: Blow up of fine grid on the nacelle upper surface.

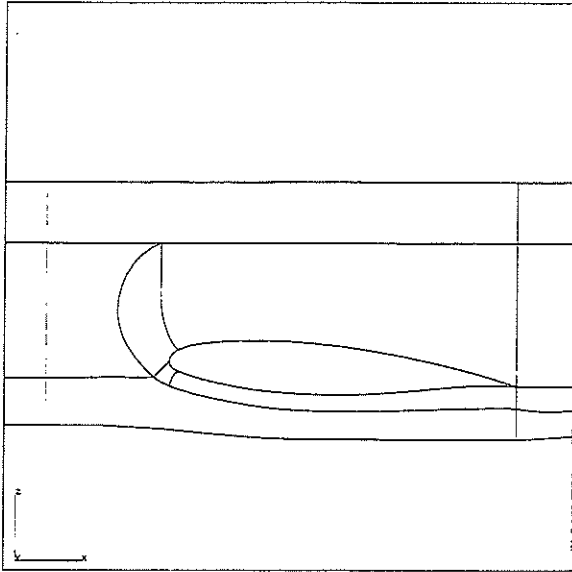


Figure 50: Block decomposition on the fuselage.

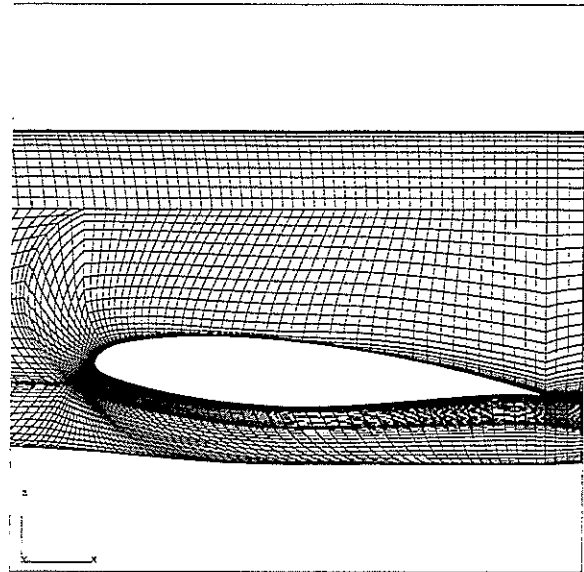


Figure 51: Fine grid on the fuselage.

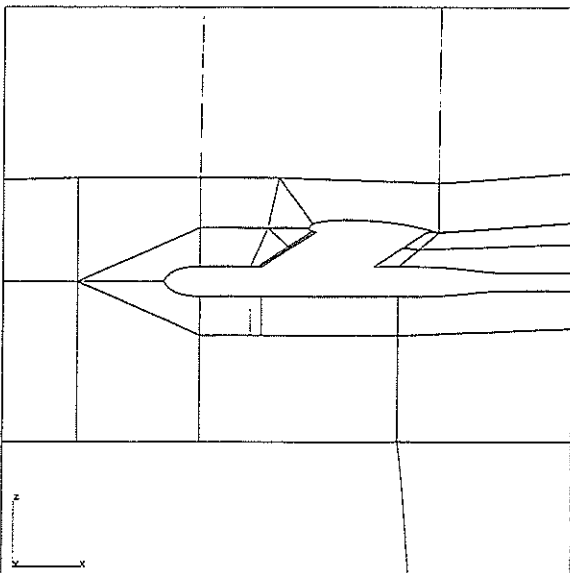


Figure 52: Block decomposition in the nacelle symmetry plane.

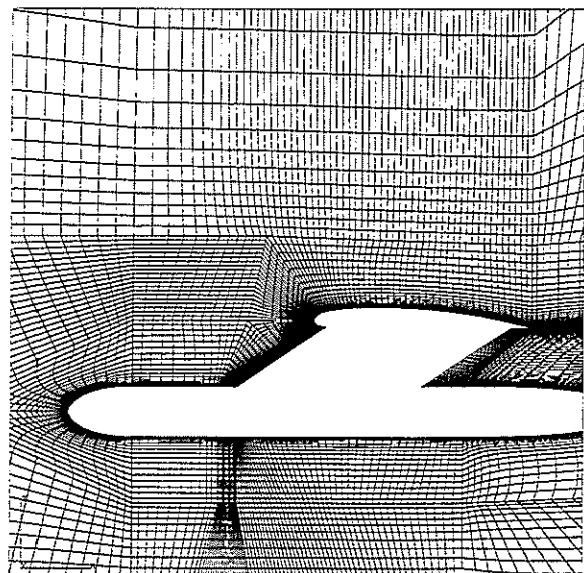


Figure 53: Fine grid in the nacelle symmetry plane.

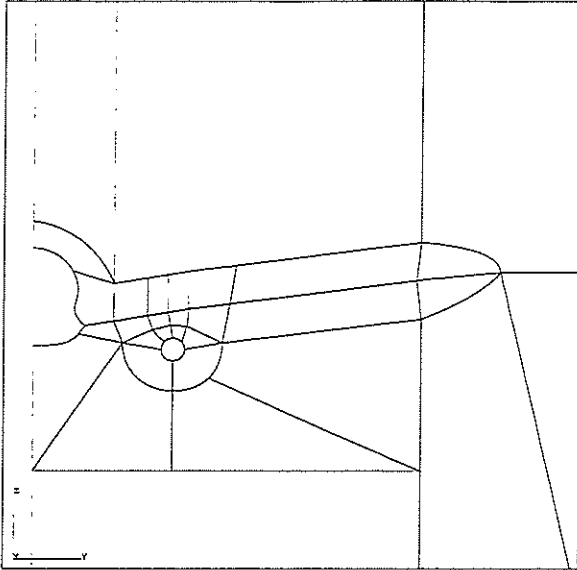


Figure 54: Block decomposition through the trailing edge of the wing.

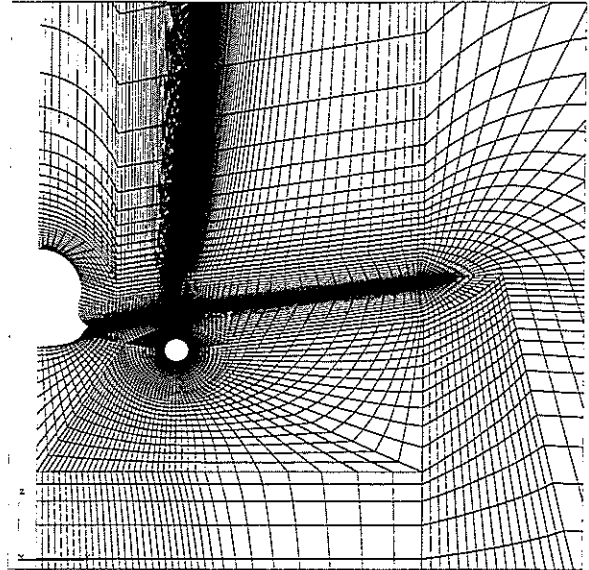


Figure 55: Fine grid in plane through the trailing edge of the wing.

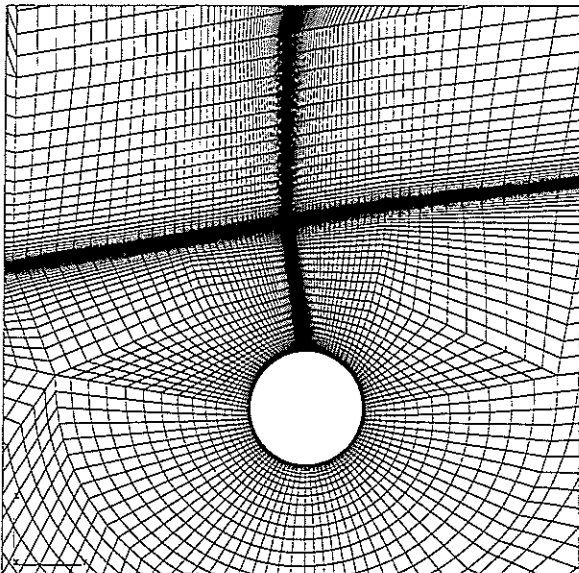


Figure 56: Blow up of fine grid near nacelle.

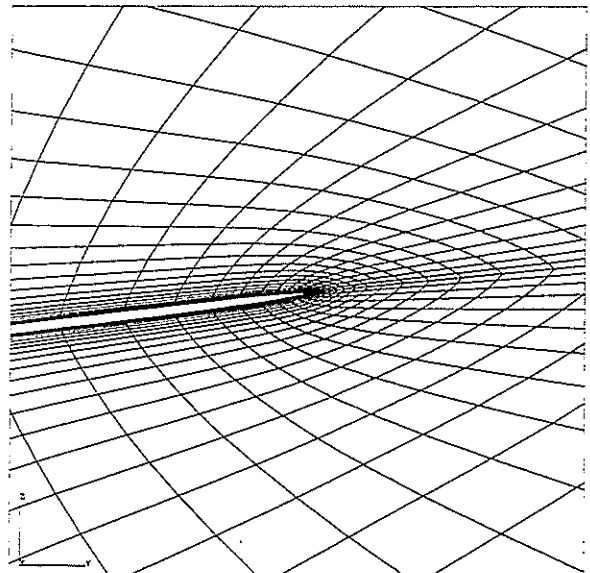


Figure 57: Blow up of fine grid near the wing tip.

