8. GRID GENERATION

Note: the terms "grid" and "mesh" are interchangeable in CFD.

Once the governing equations have been discretized, a suitable grid needs to be generated to represent the nodes and/or cells within the computational domain. In this section, we will review the various types of grids used in CFD as well as basic guidelines will be provided on how to design a grid topology.

8.1. Structured vs. unstructured grids

There are two major types of grids in CFD: structured and unstructured.

STRUCTURED:

- consists of rectangles in 2D

[n61]

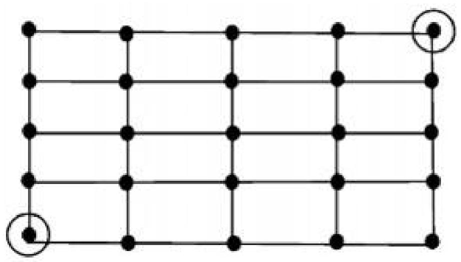


Fig. 8.1. Structured mesh.

- consists of hexahedrons in 3D

[n62]

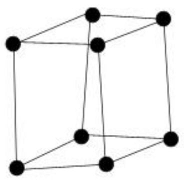


Fig. 8.3. Hexahedral element

ADV:
- connectivity is clear from indexing alone, i.e.

- easy to manipulate and store in solver

UNSTRUCTURED:

- consists of triangles in 2D

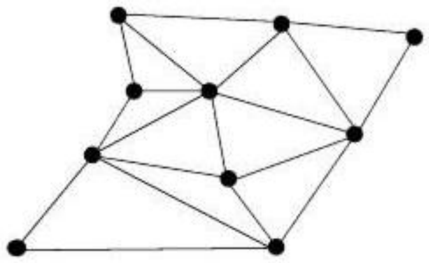


Fig. 8.2. Unstructured mesh.

- consists of tetrahedrons in 3D

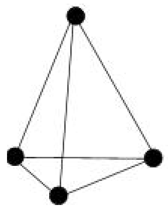


Fig. 8.4. Tetrahedral element.

DIS:
- connectivity info for each cell needs to be stored
- difficult to store and manipulate in solver

DIS:
- restrictions on orthogonality & aspect ratio
- difficult to wrap around complex geometries

- less efficient for locally refined meshes

[n63]

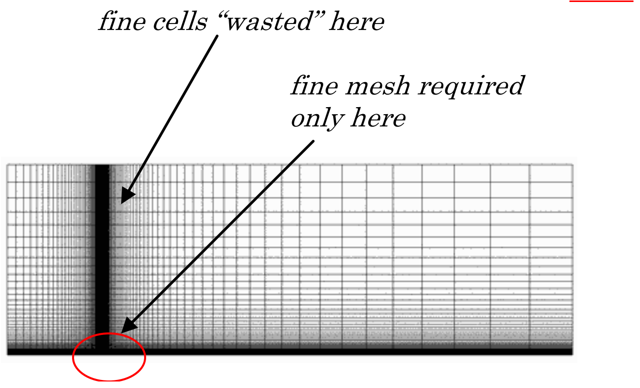


Fig. 8.5. Structured mesh.

- mesh generation: DIFFICULT (manual)

- only possibility for FD, popular for FV

ADV:
- easier to wrap around complex geoms.

- very efficient for locally refined meshes

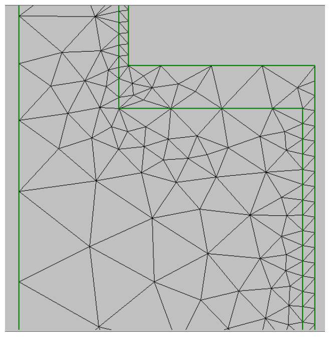


Fig. 8.6. Unstructured mesh.

- mesh generation: EASY (usually automatic)
- preferred for FE, SM, and popular for FV

8.2. Grid transformation

For most practical problems, the body geometry is usually curved and the flow exhibits regions of strong gradients. To accommodate these effects, the grid needs to be:

- **boundary-fitted:** grid wrapped around the body, e.g. around airfoil [n64]

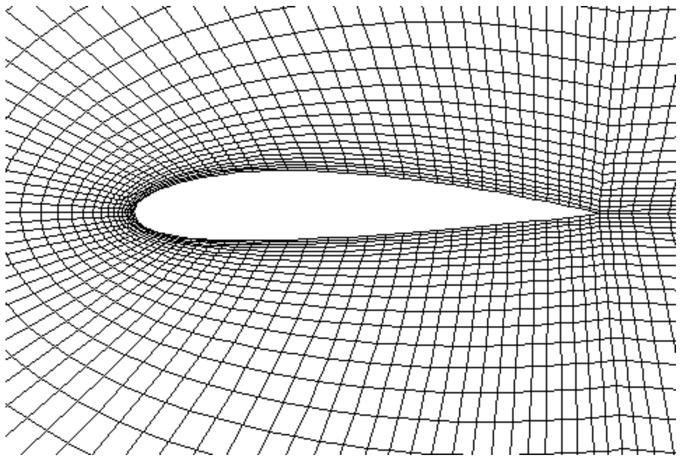


Fig. 8.7. Boundary-fitted structured mesh around an airfoil.

- **curvilinear:** cells are "rotated" relative to the x-y coordinates [n65]

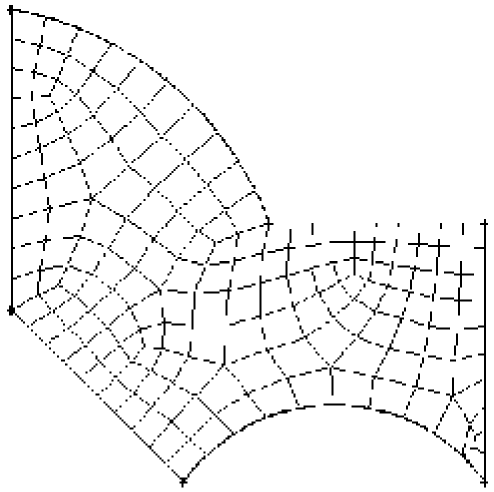


Fig. 8.8. Illustration of a curvilinear structured mesh.

- **non-uniform:** unequal spacing of cells in any one dimension [n66]

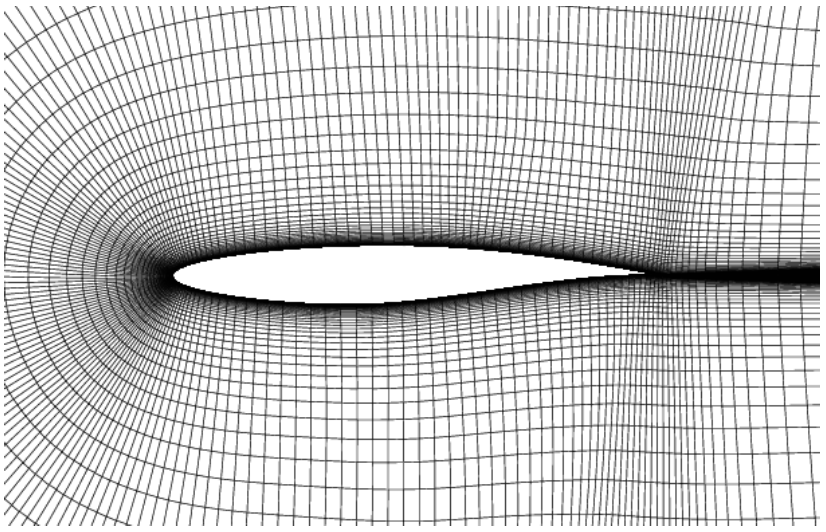
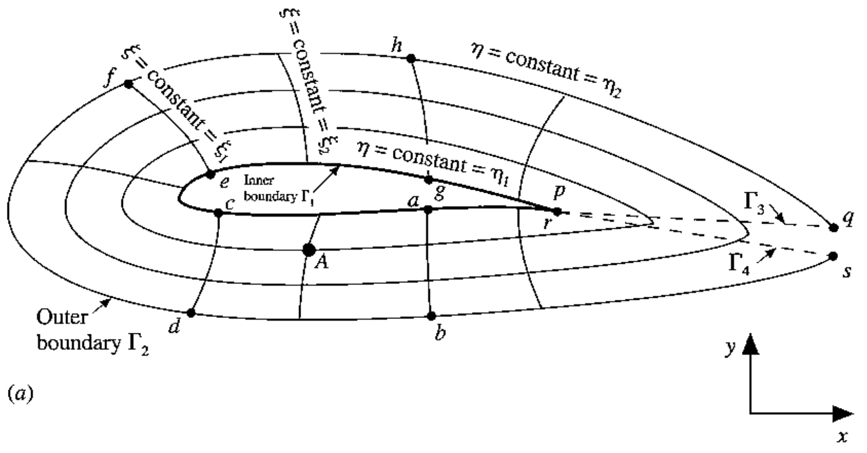


Fig. 8.9. Curvilinear, body-fitted, non-uniform structured mesh around an airfoil.

In such cases, the conventional finite difference quotients, $\frac{\partial f}{\partial x} = \frac{\Delta f}{\Delta x}$, are difficult to express and hence, the grid has to be transformed from the

physical plane

curvilinear
non-uniform
x–y coordinates


$\xi = \text{constant} = \xi_1$
$\xi = \text{constant} = \xi_2$
$\eta = \text{constant} = \eta_2$
$\eta = \text{constant} = \eta_1$
Inner boundary Γ_1
Outer boundary Γ_2
Γ_3
Γ_4
f
h
e
c
a
g
p
r
q
s
A
d
b
y
x
(a)

to the

computational plane

rectangular
uniform
ζ–η coordinates

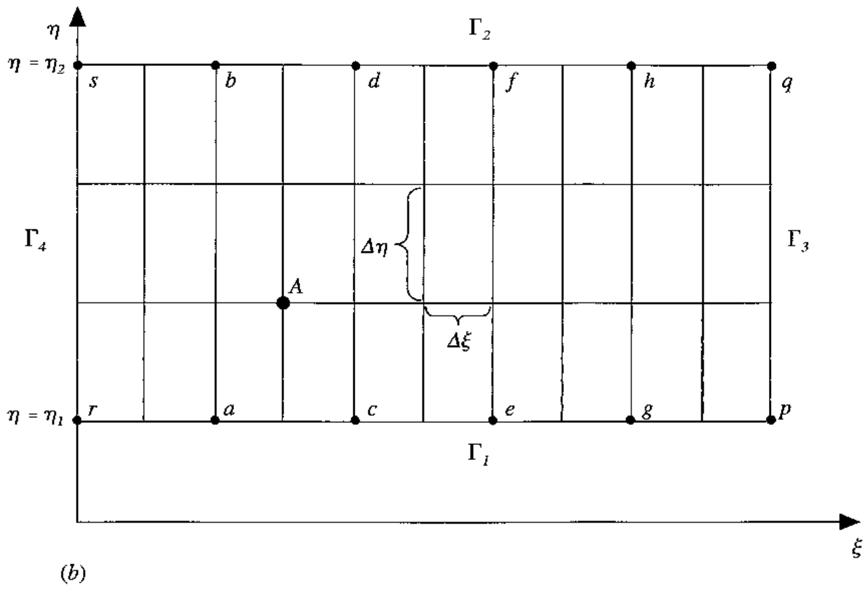


Fig. 8.10. Illustration of a grid transformation around an airfoil.

i.e. we need to transform the independent variables (*ρ, u, v, p, t,* etc) from the physical space (*x, y, t*) to the computational space (*ζ, η, τ*), where:

$$\zeta = \zeta\,(x, y, t)$$
$$\eta = \eta(x, y, t)$$
$$\tau = \tau(t)$$

In essence, we need to replace all 1st and 2nd order derivatives in the governing equations by:

[n69]

$$\left(\frac{\partial}{\partial x}\right) = \left(\frac{\partial}{\partial \xi}\right)\left(\frac{\partial \xi}{\partial x}\right) + \left(\frac{\partial}{\partial \eta}\right)\cdot\left(\frac{\partial \eta}{\partial x}\right)$$

$$\left(\frac{\partial}{\partial y}\right) = \left(\frac{\partial}{\partial \xi}\right)\cdot\underbrace{\left(\frac{\partial \xi}{\partial y}\right)}_{\text{metric term}} + \left(\frac{\partial}{\partial \eta}\right)\cdot\underbrace{\left(\frac{\partial \eta}{\partial y}\right)}_{\text{metric term}} \quad \text{etc.}$$

etc... (usually obtained by central diff.)

$$\left(\frac{\partial^2}{\partial x \partial y}\right) = \text{- - - - - - long-long eq. - - -} \quad \text{see And. 171-186}$$

Using these expressions makes the governing equations in the computational space (*ζ, η, τ*) to look awkward and way too involved mathematically. So, a more convenient form of the transformation is obtained by using *inverse metrics* or *Jacobians*:

Inverse metrics:
[n70]

$$\left.\begin{aligned}\frac{\partial u}{\partial \xi} &= \frac{\partial u}{\partial x}\frac{\partial x}{\partial \xi} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial \xi}\\ \frac{\partial u}{\partial \eta} &= \frac{\partial u}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial \eta}\end{aligned}\right\}$$

Note that this a system of 2 equations with 2 unknowns, $\partial u/\partial x$ and $\partial u/\partial y$.

We can find its solution by using Cramer's rule:
[n71]

$$\frac{\partial u}{\partial x} = \frac{\begin{vmatrix} \frac{\partial u}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial u}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{vmatrix}}{\underbrace{\begin{vmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{vmatrix}}} \Longrightarrow J \equiv \frac{\partial(x, y)}{\partial(\xi, \eta)} \equiv \begin{vmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{vmatrix}$$

Jacobian matrix
(denoted as J)

and one can then express the governing equations (in the example below the Euler equations) in the computational space as:
[n72]

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = 0 \Longrightarrow \frac{\partial U_1}{\partial t} + \frac{\partial F_1}{\partial \xi} + \frac{\partial G_1}{\partial \eta} = 0$$

with:

$$U_1 = JU$$

$$F_1 = JF\frac{\partial \xi}{\partial x} + JG\frac{\partial \xi}{\partial y}$$

$$G_1 = JF\frac{\partial \eta}{\partial x} + JG\frac{\partial \eta}{\partial y}$$

(See Anderson pp. 171-186 for full details)

8.3. Cartesian meshes

A special class of structured meshes are Cartesian meshes, which are grids with maximum degree of structure. For Cartesian meshes, *all cells are square (or in 3D cube) shaped, which sides are aligned with the exes of the coordinate system* (Fig. 8.11).

Since Cartesian meshes combine the best of both worlds from the unstructured and structured grids, i.e.

- easy mesh generation (typical for unstructured grids)
- easy storage and manipulation of cell connectivity (typical of structured meshes)

they have become popular recently (since about 2000) for complex and large scale problems where frequent re-meshing is required during the calculation (such as predicting the near-aircraft path of a missile launched from an F-18 fighter jet).

It is really the evolution of the Finite Volume method, which opened the avenue for this type of grids, since the *truncated cells near the surface can be easily solved by Finite Volume methods* (for which the cell shape is not important).

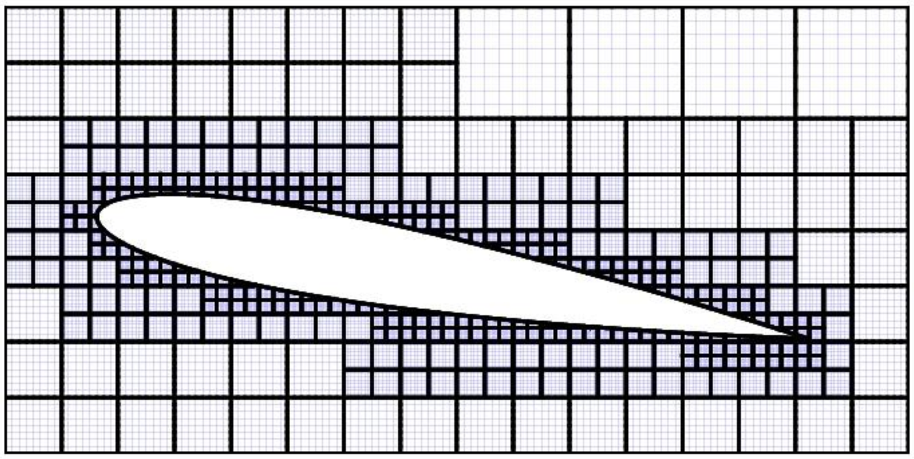


Fig. 8.11. Cartesian mesh around an airfoil. Note that the cells are locally refined around the wall.

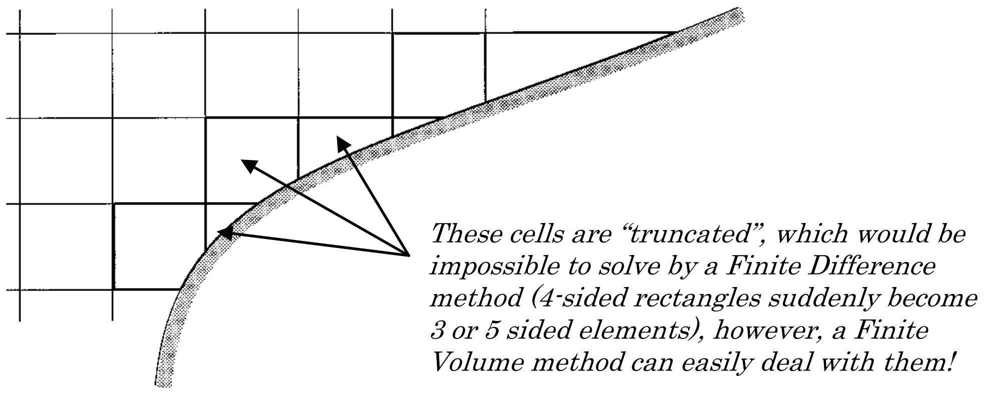

These cells are "truncated", which would be impossible to solve by a Finite Difference method (4-sided rectangles suddenly become 3 or 5 sided elements), however, a Finite Volume method can easily deal with them!

Fig. 8.12. Cartesian mesh around the boundary of an airfoil. Note that cells around the boundary are cut and therefore are not four-sided any more.

Cartesian meshes are used for very complex geometries (full F-18 fighter jet model with extended landing gears, missiles and auxiliary fuel tanks at wing tips) where grid generation with other techniques would take ages. Combined with *"adaptive multigrid"* techniques (to be discussed later), it forms a very powerful tool, although often limited to inviscid flows only. This is the method employed for example the commercial VECTIS code as well.

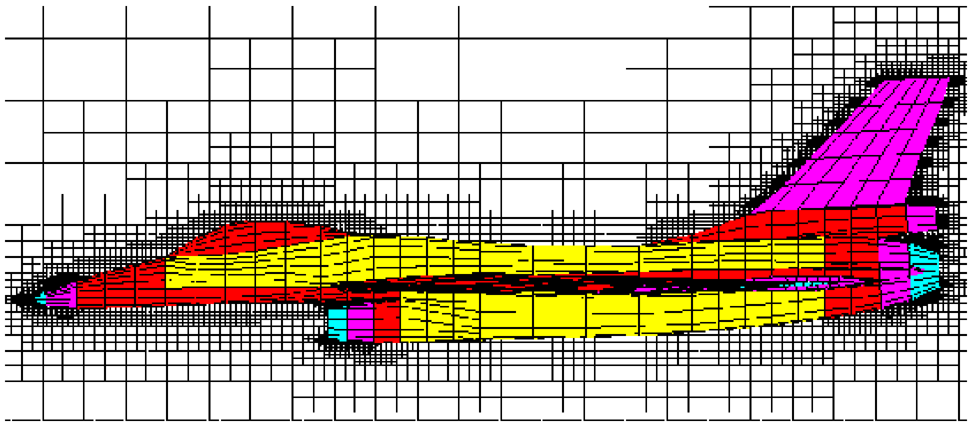
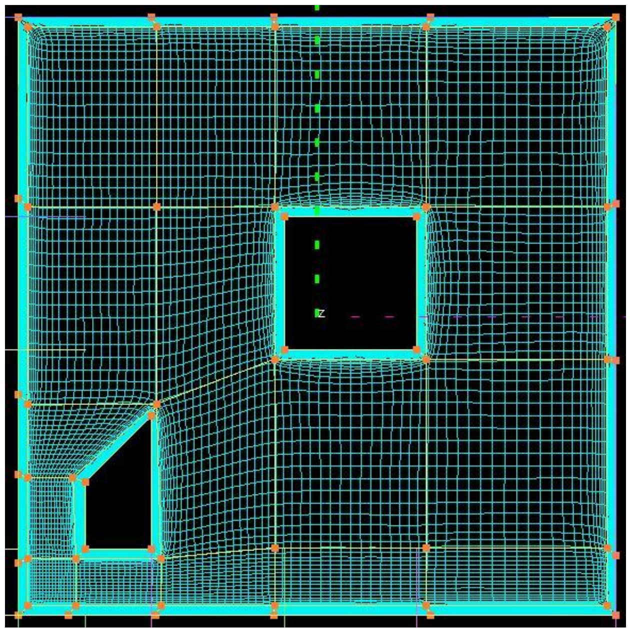


Fig. 8.13. Cartesian mesh around and F-16 aircraft.

8.4. Zonal or Block-structured grids

Many solvers are set up to work with meshes divided into zones (the term frequently reserved for unstructured meshes) or blocks (the same term used for structured meshes):

STRUCTURED MESH



UNSTRUCTURED MESH
(each colour represents a zone)

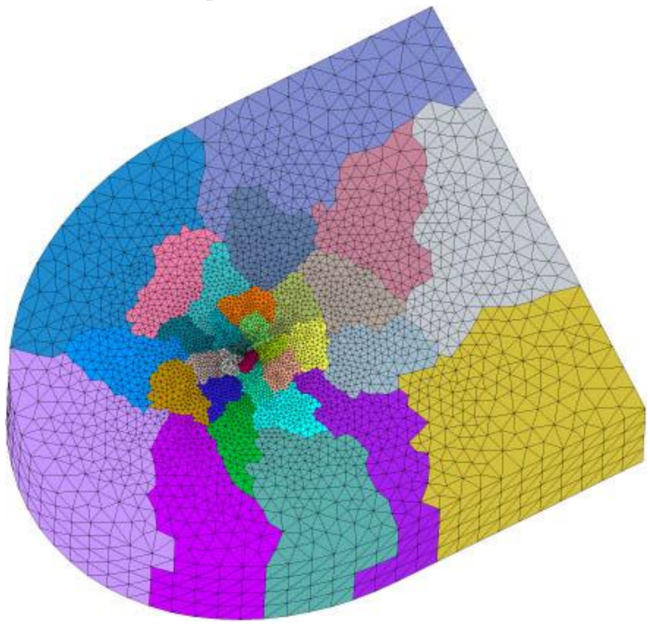


Fig. 8.13. Multi-block structured mesh (left) and unstructured zonal mesh (right). The different rectangles/trapezoids on the left and the different colors on the right represent different blocks (or zones).

Introducing zones/blocks has the following main advantages:

- easy control over local mesh refinement
- suitability for parallel processing (each zone/block run on a different processor)

The implications for a zonal/blocked mesh are:

- connectivity of zones/blocks has to be defined/stored in the solver
- boundary conditions for a block face is read from the neighbouring block's corresponding face
- each block/zone face must form one type of boundary, i.e. mixed boundary conditions on any one face are typically not allowed:

[n74]

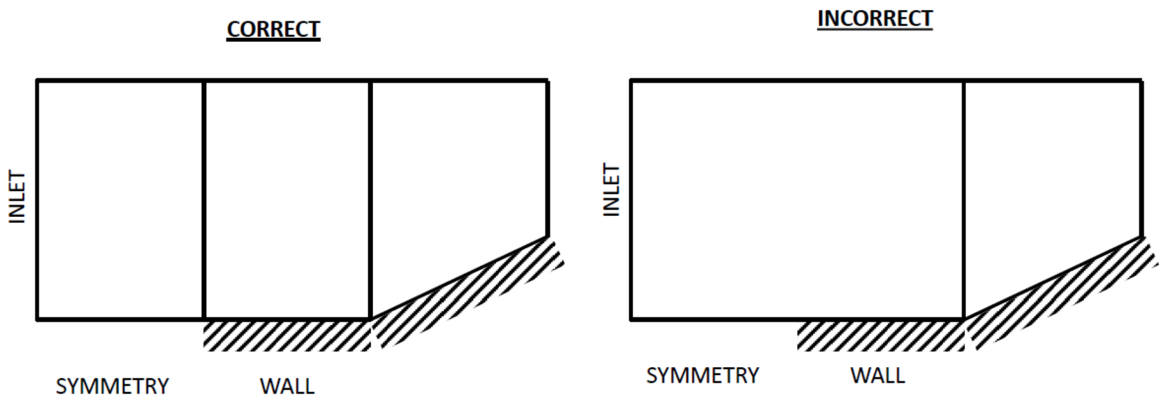


Fig. 8.14. Rules for block connectivity: an edge of a block usually must be of the same boundary type!

- for many block-structured solvers, only compatible block sizes are allowed, i.e.

[n75]

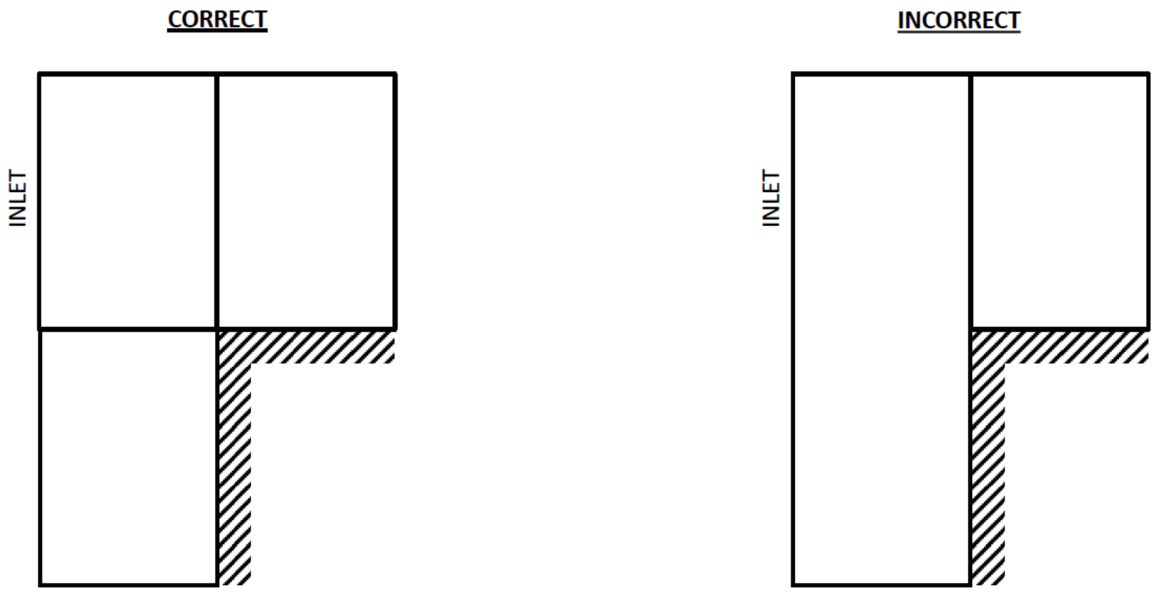


Fig. 8.15. Rules for block connectivity: only compatible block sizes are allowed!

8.5. Hybrid meshes

To capture boundary layers in viscous (laminar or turbulent) simulations, one needs to introduce a quite a fine mesh near solid surfaces. The general rule is:

- to have at least 10-15 points inside the boundary layer

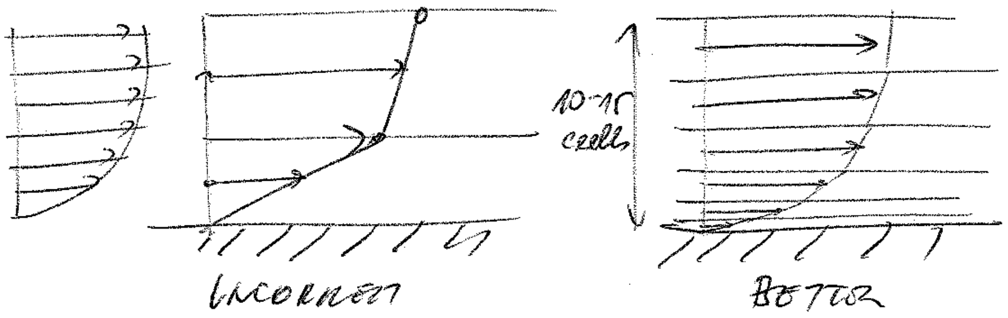


Fig. 8.16. In order to resolve a boundary layer properly, at least 11-15 cells need to be inside the boundary layer.

- the first spacing near the wall shall be in the order of *$y_P^+ = (1\sim10)$*, (i.e. inside the viscous sublayer) for models without wall functions.

- the first spacing near the wall shall be in the order of *$y_P^+ > 30$*, (i.e. outside the viscous sublayer) for models with wall functions

Achieving such grid density near the wall is often not efficient - the node count is typically much higher than for structured meshes. Hence, many solvers, such as ANSYS-CFX, offer the possibility of using hybrid meshes, which combine the advantages and disadvantages of structured and unstructured meshes, i.e.

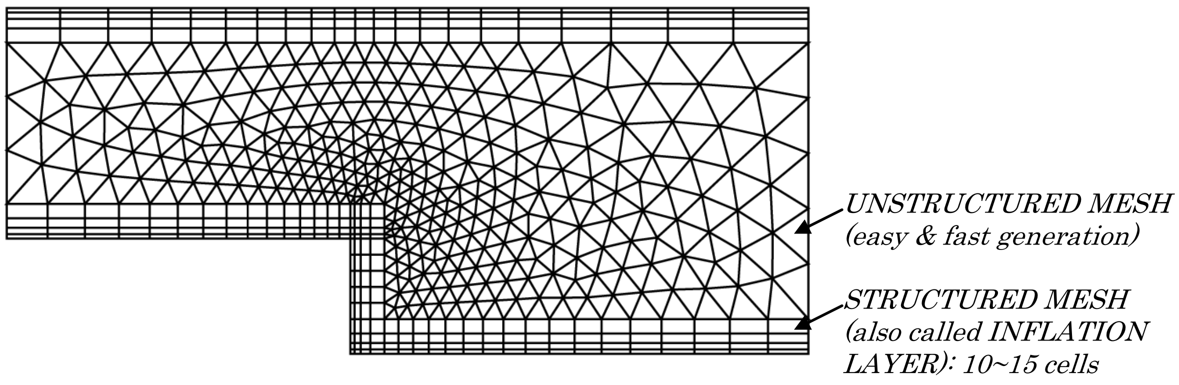


Fig. 8.17. Hybrid structured-unstructured mesh. Structured mesh is used around the solid wall boundaries to capture the boundary layer, while unstructured mesh in the freestream. This is the method employed by CFX and Fluent in ANSYS.

Note that in 3D, a hybrid mesh calls for PRISM type cells in the inflation layer and TETRAHEDRON type cells elsewhere.

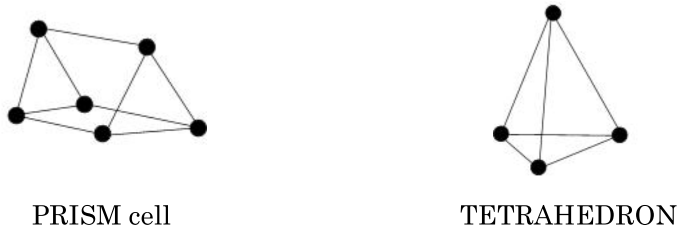


Fig. 8.18. Cells created in a hybrid structured-unstructured mesh. The PRISM cell occurs in the structured mesh part, while the TETRAHEDRON in the unstructured mesh part of the domain.

8.6. Moving mesh techniques

Many applications require that the body of interest moves across the stationary computational domain. Examples involve releasing a missile from an aircraft, tracking the debry falling off from the Space Shuttle, rotating blades of a helicopter rotor, etc.

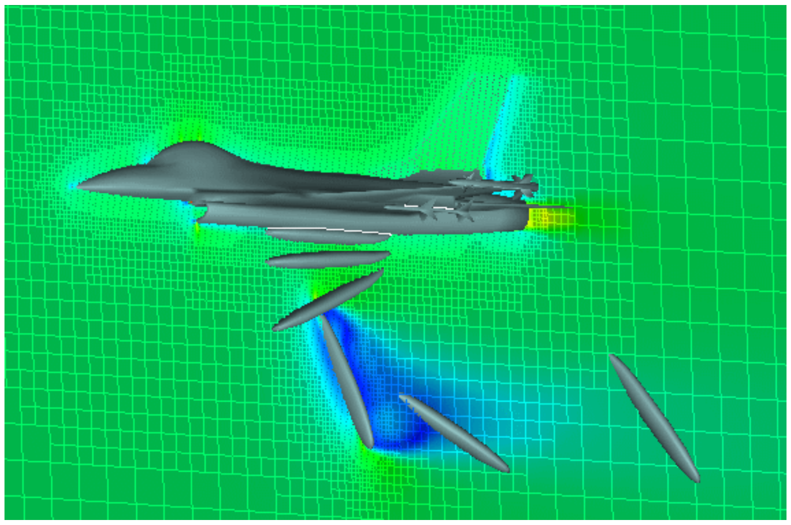


Fig. 8.19. CFD simulations of releasing a missile from an F-16 aircraft. Such simulations require a moving mesh technique to capture the missiles trajectory across the stationary computational domain.

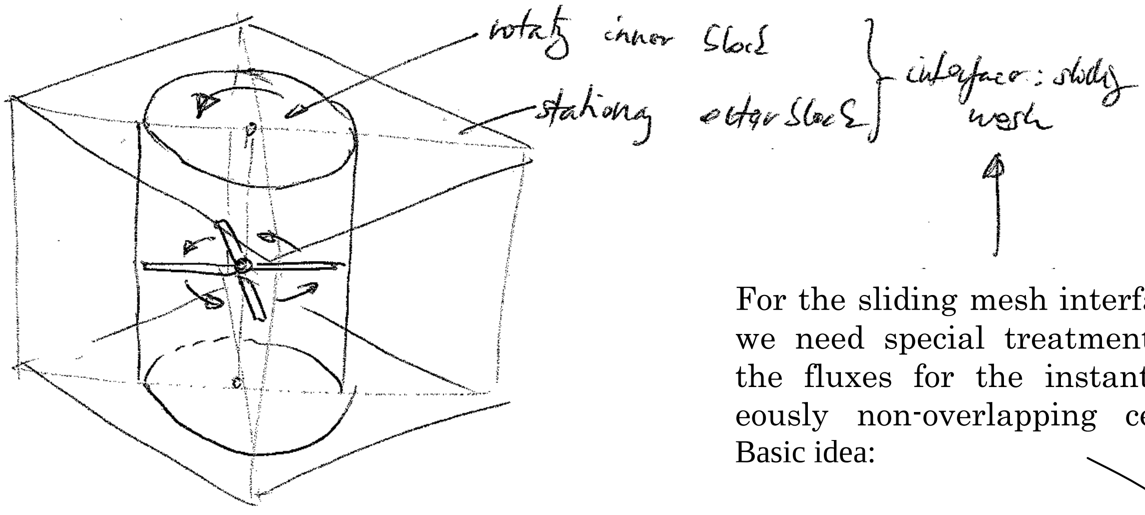
For such cases, various moving mesh techniques need to be applied, from which we will review the 2 most popular techniques:

- the Sliding Mesh technique
- the CHIMERA technique

8.6.1. Sliding mesh technique

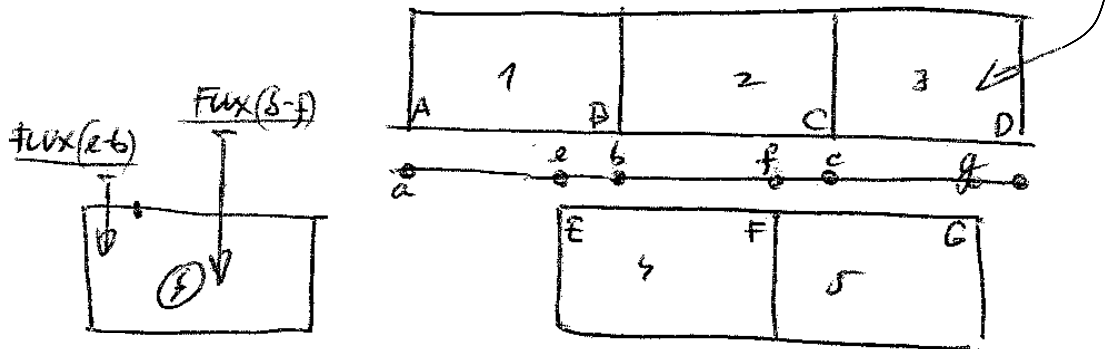
Commonly used for problems, where the motion of the body is not arbitrary. This is a very popular technique for rotating bodies, such as turbines, impellers, propellers, or helicopter rotors.

[n76]



For the sliding mesh interface, we need special treatment of the fluxes for the instantaneously non-overlapping cells. Basic idea:

[n77]



Total flux for cell (4) is calculated as the sum of the weighted fluxes from sections (e-b) and (b-f).

Fig. 8.20. Sliding meshes are used for simulating geometries moving inside a stationary domain.

8.6.2. CHIMERA technique

Note: CHIMERA is word originating from Greek mythology, representing a hybrid animal composed of parts of other animals, such as a lion and a deer.

Fig. 8.21. A Chimera is a creature in Greek mythology, composed of parts from different animals.

In CFD, the CHIMERA technique involves using 2 overlapping grids, one which is fixed to the moving body (Minor Grid), and one, which is fixed in the coordinate system (the Major Grid). Then, the flow variables are calculated with a relatively complex interpolation technique between the two grids.

CHIMERA mesh: Interpolation required in the overlapping cells after each iteration and movement of body.

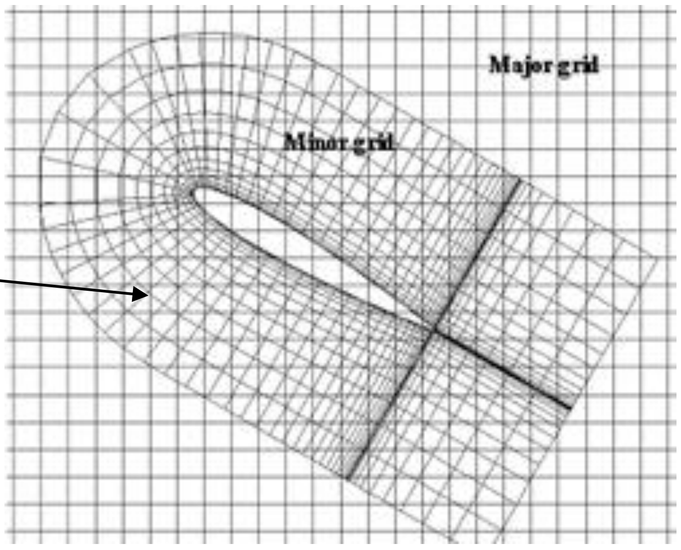


Fig. 8.22. A Chimera mesh for an airfoil simulation with time-varying angle of attack.

CHIMERA is ideal for problems where the trajectory of the moving body is not known *a priori*. An example is the missile release from an F-16, in which the missile has 6 degrees of freedom movement.

Note that one also requires a mechanism to cut-off or discard those cells of the Major Grid which are covered by the moving body.

8.7. Deforming Mesh techniques

When the body of interest moves only slightly relative to the computational domain, it is possible to deform the cells of the computational domain via an automatic regeneration technique. This is called the Deforming Mesh method. One example is the automatic regeneration of the mesh based on the wall surface motion, for which for example the so-called Trans-Finite-Interpolation (TFI) technique can be used.

[n78]

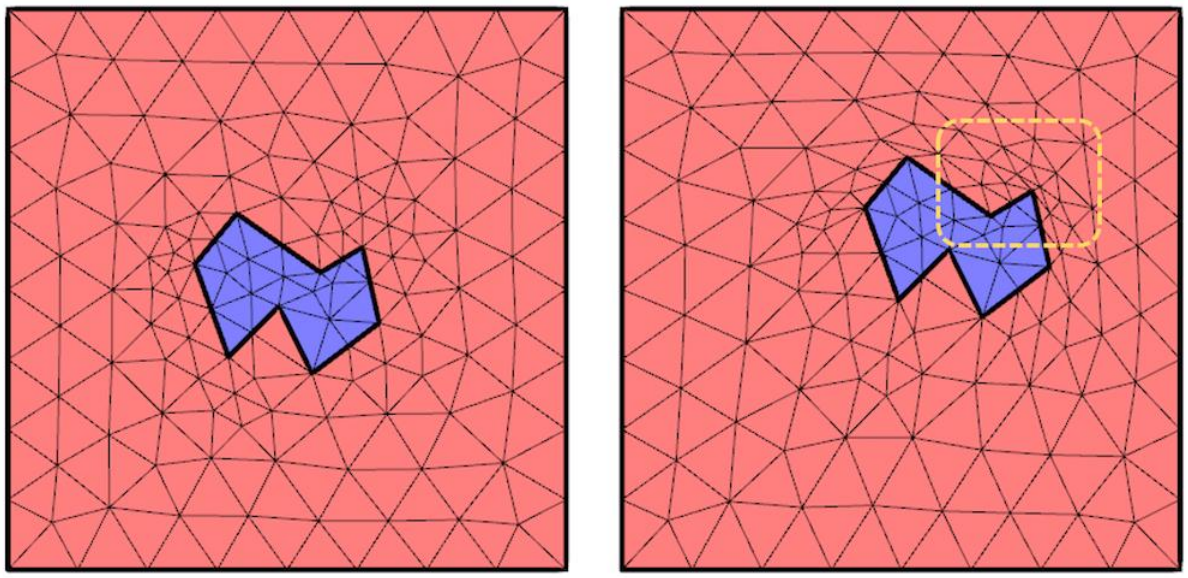


Fig. 8.23. Deforming or "dynamic" mesh. Note that the total number of cells has not changed but their topology has been deformed in the pink zone in oder to accommodate the motion of the purple zone.

8.8. Adaptive mesh

An adaptive mesh is a grid network that automatically clusters grid points in regions of high gradients. It evolves in steps of time in conjunction with a time dependent solution of the governing equations. Adaptive meshing usually means redistribution of the existing cells, instead of introducing new ones.

ADV:
- no need to know the flow details (such location of shock waves, thickness of boundary layer, etc.) in advance.
- increased accuracy in comparison to fixed-grid approaches

DIS: - grid regeneration increases computational cost.

Example:

$$\Delta x = \frac{B \cdot \Delta \xi}{1 + b\left(\dfrac{\partial g}{\partial x}\right)}$$

with: g…. one of the variables (*ρ, u, v, w, p, e, T, etc.*)
B…. scaling factor
b…. weighting factor, representing effect of gradient on the grid resolution

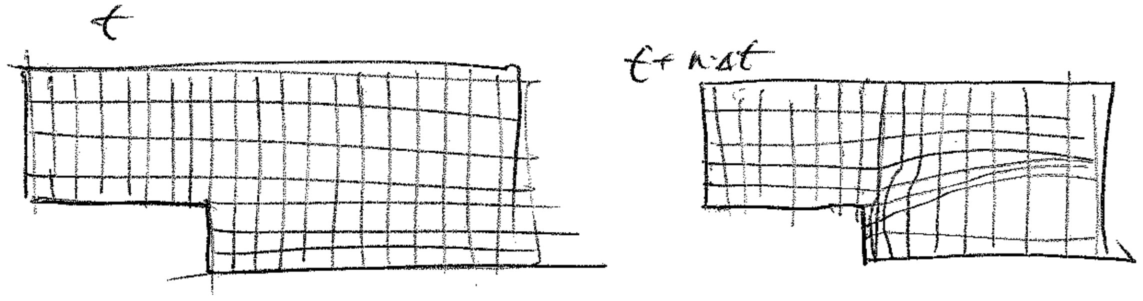


Fig. 8.23. Adaptive mesh technique automatically clusters cells around the region of high pressure gradients.

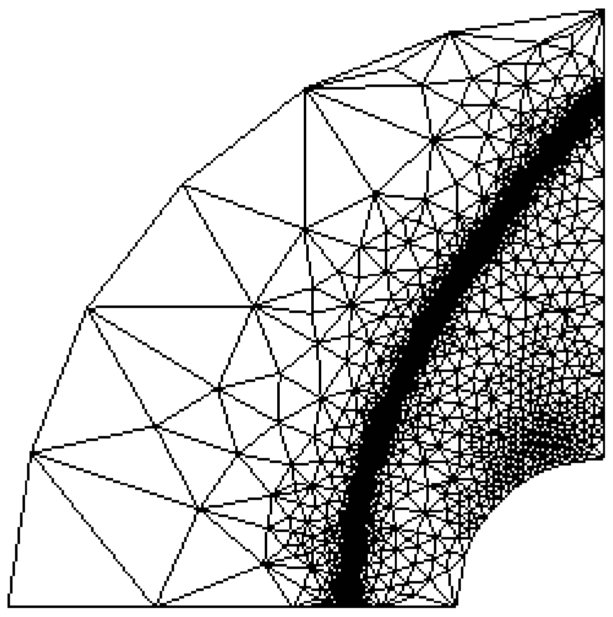


Fig. 8.24. Adaptive mesh technique applied to high speed flow over a blunt body. Note that most of the cells have been automatically clustered by the code around the region of high pressure gradient, i.e. the shock wave! This would yield nice sharp shock wave.

Adaptive meshes are relatively new t g
for compressible flows with shock wa

8.9. Multigrid

This is rather a topic related to the efficient solution of large systems of equations, however, since it has some implication on grid generation, it will be discussed here briefly.

The philosophy of the multigrid method is to gain superior convergence rates by carrying out the early iterations on a fine grid and then to progressively transfer these results onto a coarse grid. The basic algorithm is the following:

1) perform a few iterations on a fine grid
2) project fine grid results onto a coarse grid
3) perform a few iterations on coarse grid
4) project coarse grid results onto fine grid
5) Go back to step 1) and repeat until solution is converged

When using a multigrid method, one has to "design" the fine grid topology such so that when it is reduced to a coarse grid – usually by successively removing every 2nd gridpoint – the grid still makes sense around the wall or around shock waves.

8.10. Guidelines for grid generation

There are 3 main rules for grid generation:

1. The size of the computational domain should be adequate for the problem (not too small, not too large)

2. Cell numbers & cell density should be adequate for the problem

3. Always "design" your mesh on paper (think about boundarys, blocks arrangement, grid spacing around walls)

We will go through these 3 points on the next few pages.

1) Size of computational domain

SUBSONIC FLOWS: Since in subsonic flows, information propagates upstream (see section 4.6.), the domain has to be large enough to "give space" to these waves. This is especially true for unconfined flows (aerospace):

[n79]

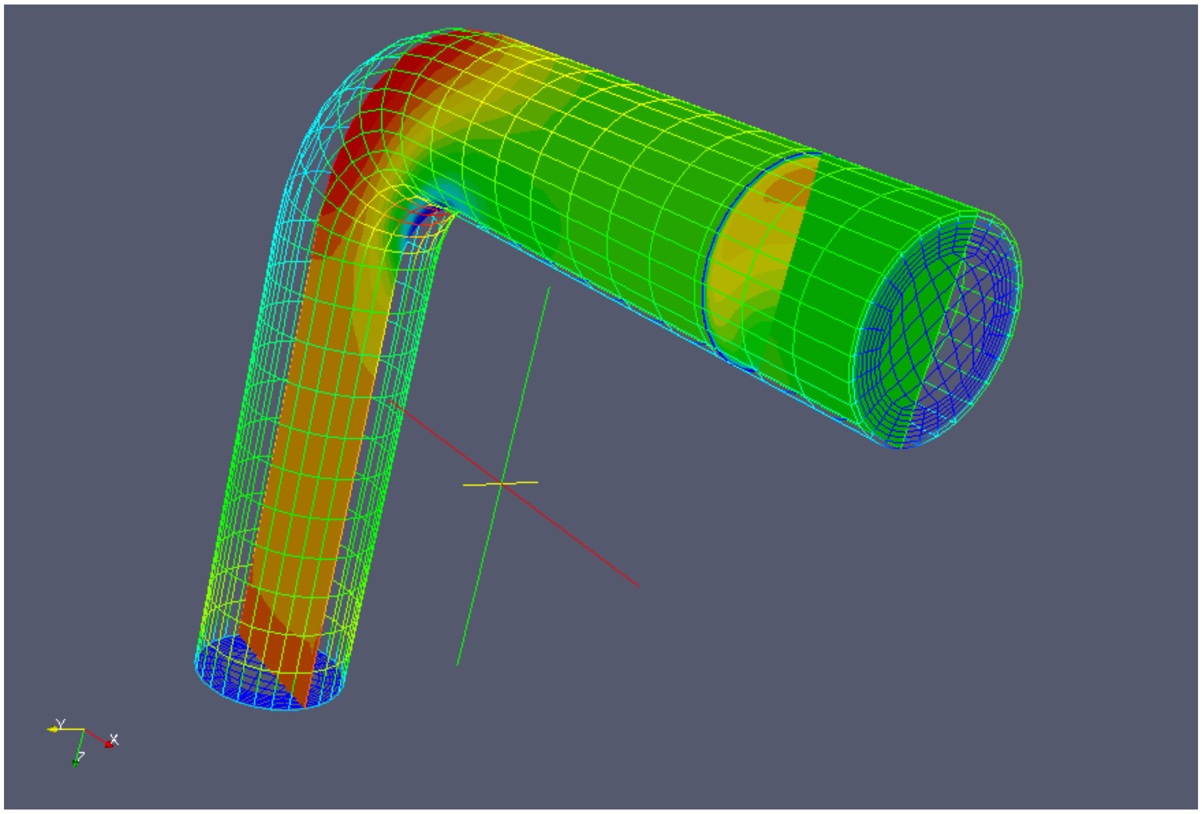
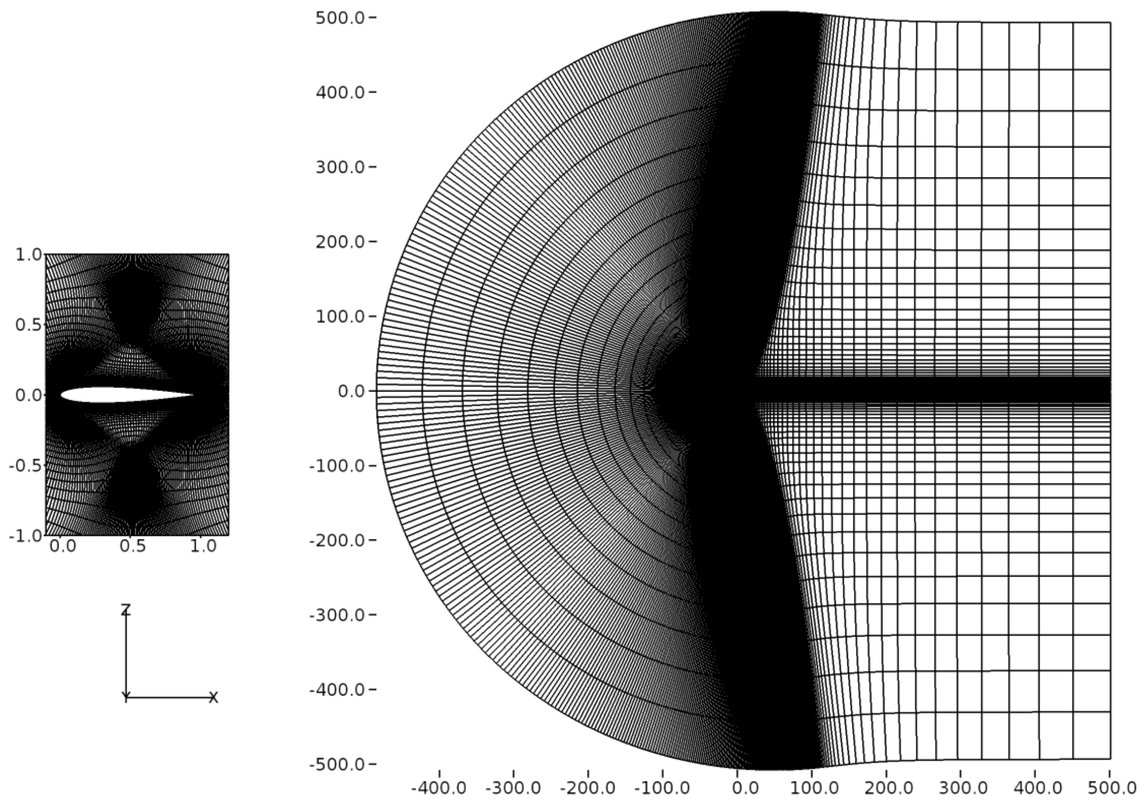


Fig. 8.25. For internal flows, the size of the computational domain is defined (confined) by the edges of the geometry.



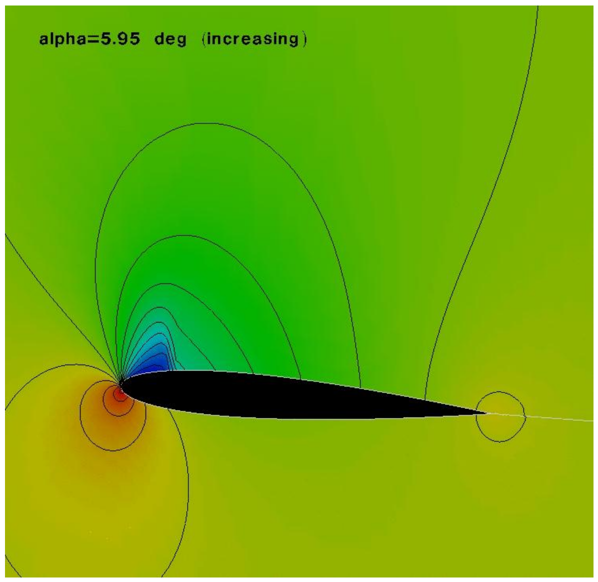
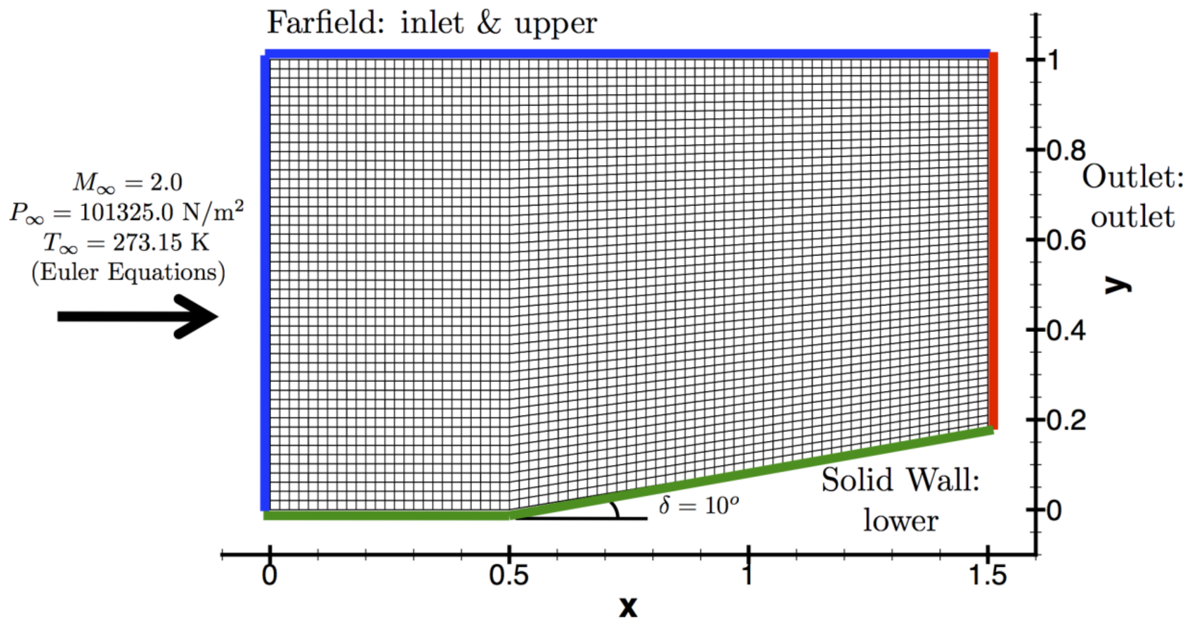


Fig. 8.26. For external subsonic flows, the size of the computational domain (top) must be large enough to allow information (pressure waves: bottom) to propagate and develop in all directions. Note that in this particular case, the domain edges are as much as 500 times the airfoil chord length (which is 1.0) away from the airfoil. Typically, a minimum of 10 chord length is required, but the more, the better.

SUPERSONIC FLOW: Since information can propagate downstream only, the goal is to set the upstream boundaries as close as possible to the shock wave. Make sure that the shock wave exits the domain at an "outlet" boundary (to be discussed in the next chapter), instead of a "farfield" boundary.
[n80]



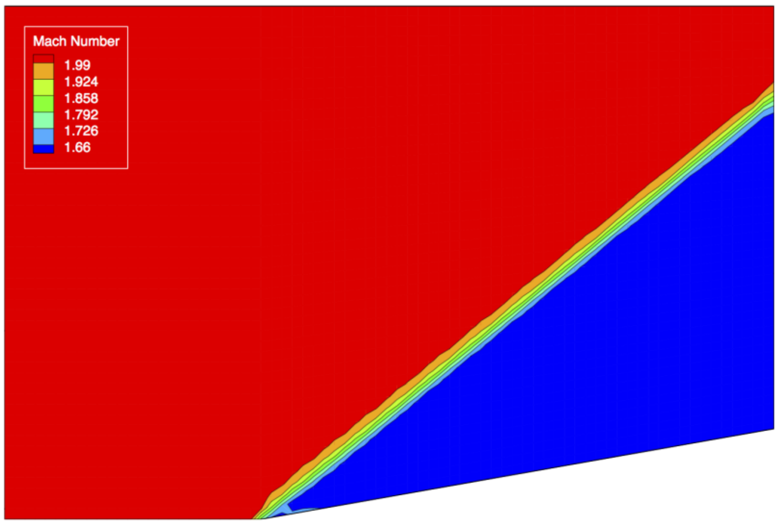


Fig. 8.27. For external supersonic flows, the size of the computational domain can be small since information (pressure waves) propagate only downstream. Mesh (top) and Mach number contours (bottom) for a shock wave forming around a 10 degrees wedge at Mach 2.

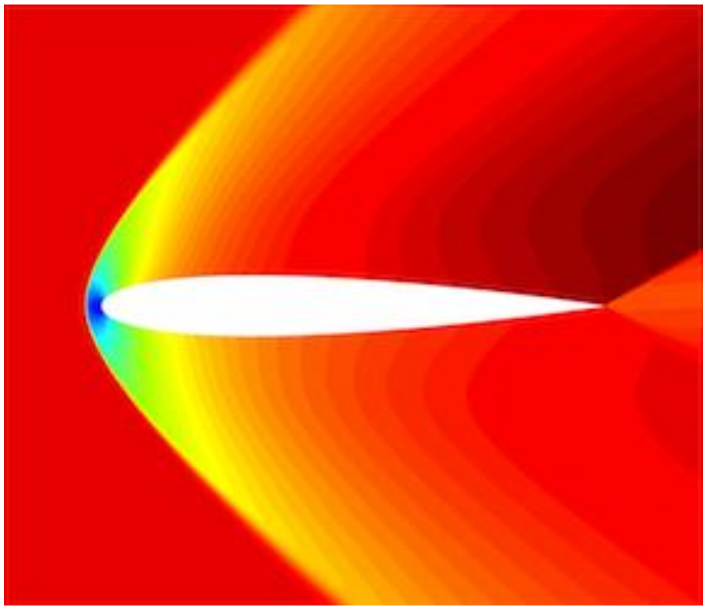


Fig. 8.28. Mach number contours for a supersonic airfoil. Note that the flow is undisturbed upstream of the airfoil because information can propagate only downstream. This allows for a smaller computational domain than in Fig. 8.26 (domain not shown).

FLAT PLATE: Flow problems involving the development of a boundary layer over a flat plate should incorporate and extra "entry block", which allows the proper development of the boundary layer. Without this block, singularity problems could arise at the 1st point of the flat plate (the corner cell has a no-slip condition from the bottom and a farfield condition from the front) and computations usually crash in the first few steps.

[n81]

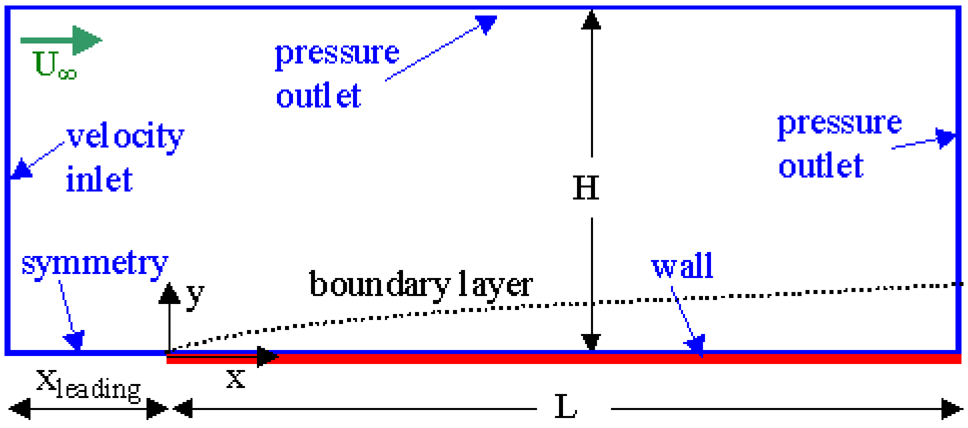


Fig. 8.28. Comptatational domain for a flat plate.

GRID TYPE: For airfoils, there are several common grid topologies used, such as O-grid, C-grid, C-H grid, etc. A few examples are shown below.

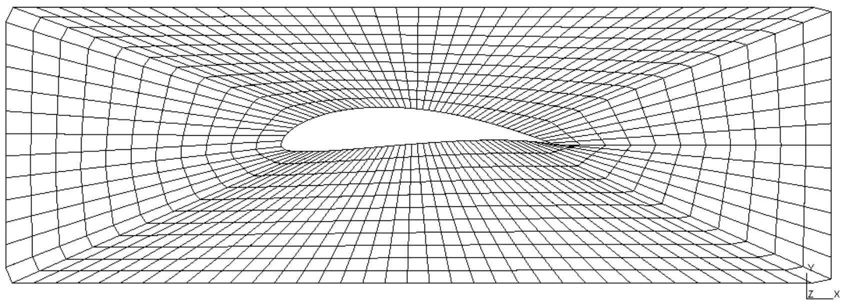


Fig. 8.29. Topology of an O-grid.

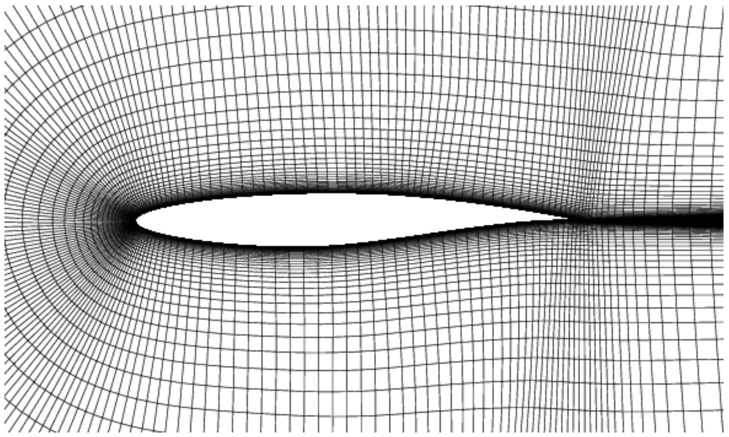


Fig. 8.30. Topology of a C-grid.

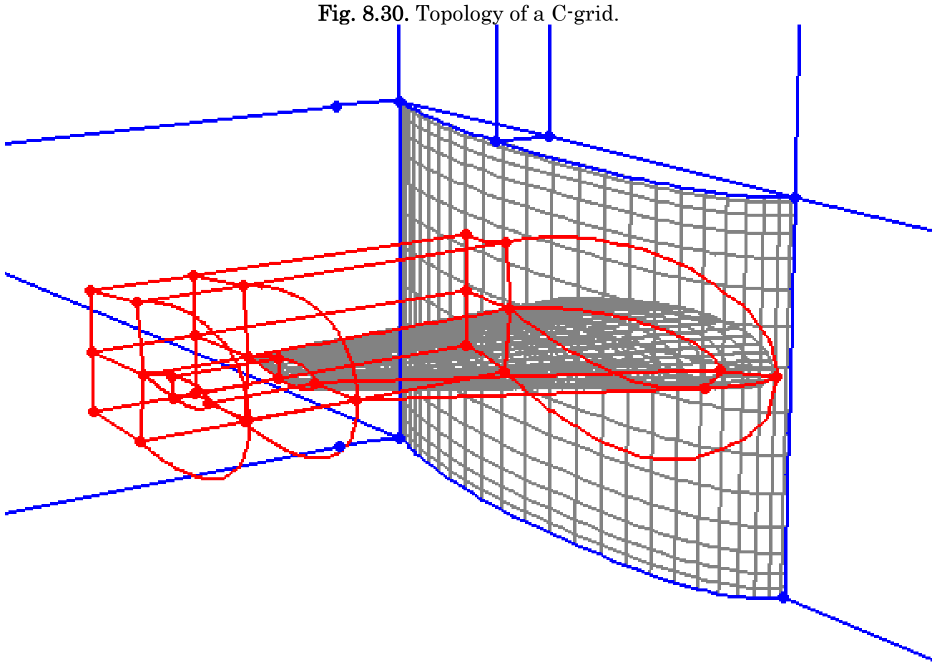


Fig. 8.31. Topology of a C-H grid.

2D MESHES: In a 2D solver, the grid is simply planar (i.e. existing in the x-y plane). In a 3D solver (such as ANSYS-CFX) 2D cases are solved by using symmetry conditions on the front and aft planes. For these cases, it is recommended that the domain is 1 cell wide in the z-direction for structured meshes with HEXAHEDRON elements. For unstructured meshes, try to use the PRISM cells (see Sec. 8.5), or if this is not allowed by the grid generator, use 2-4 cells of TETRAHEDRONS.
[n83]

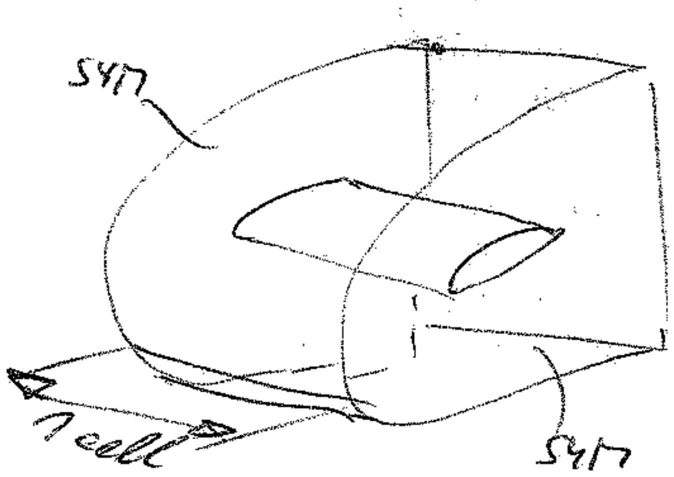


Fig. 8.32. 3D computational domain with only 1 cell in the z-direction for a 2D airfoil simulation.

AXISYMMETRIC CASES: In a 2D solver, usually a choice of 2D (planar) or axisymmetric simulation can be selected. For the axisymmetric choice, the governing equations are extended with extra terms accounting for the axisymmetric terms. These account for "radial flow effect", which effect is that for example the shock detachment distance over the same 2D grid is going to be less for an axisymmetric run than for a 2D (planar) run.
[n84]

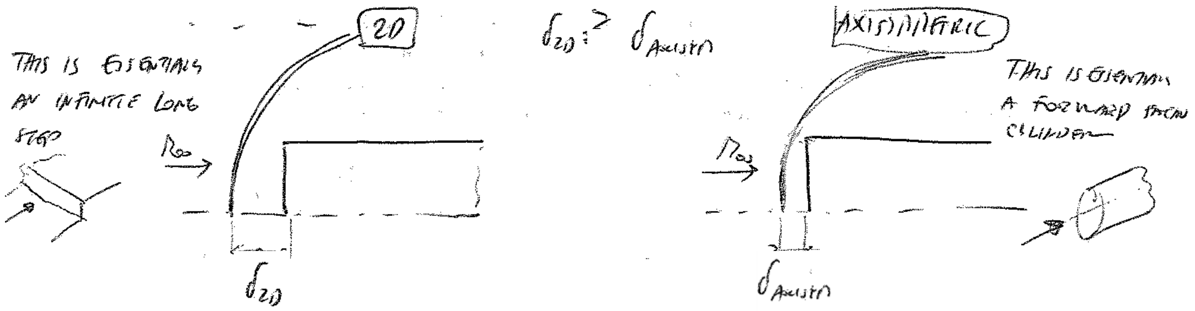

THIS IS ESSENTIALLY AN INFINITE LONG STEP
M_∞
2D
δ_{2D}
$\delta_{2D} \overset{?}{>} \delta_{Axisym}$
AXISYMMETRIC
M_∞
THIS IS ESSENTIALLY A FORWARD FACING CYLINDER
δ_{Axisym}

Fig. 8.33. 2D (left) and axisymmetric (right) meshes. The double lines show the expected location of shock waves in supersonic flow. The shock detachment distance in supersonic flow will be always less for the axisymmetric case.

2) Cell number and cell density

General rule: cluster points in region of large gradients.

GRID CLUSTERING IN BOUNDARY LAYER:

- have at least 10~15 cells in boundary layer (20 is even better)

- use continuously expanding grid in the normal direction
  - expansion ratio should be around 1.2~1.5 (many codes cannot handle larger expansion)

$$\frac{\Delta y_{j+1}}{\Delta y_j} = (1.2 \sim 1.5)$$

- use continuously expanding grid in the streamwise direction if boundary layer separation or reattachment is to be captured.

[n85]

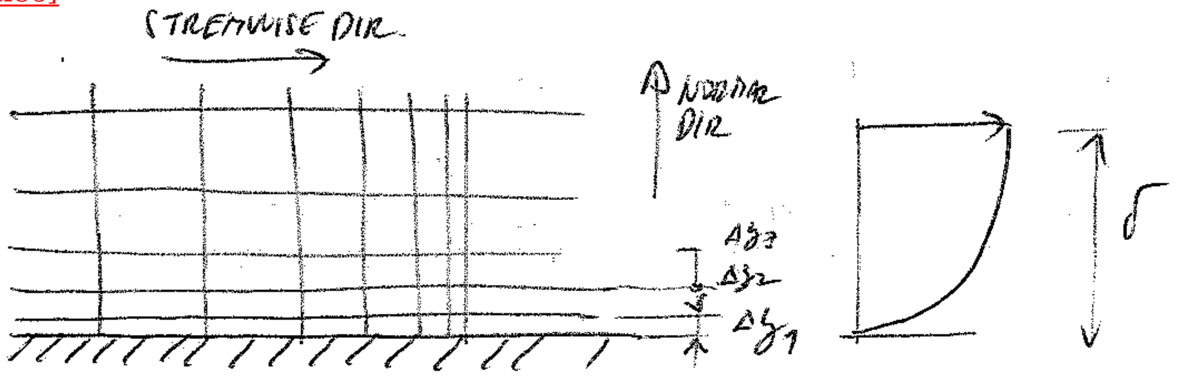


Fig. 8.34. Guidelines for setting up the mesh in a boundary layer.

- first spacing around wall:
  - LAMINAR B.L.: based on feeling and requirement to fit 15 cells into B.L.
  - TURBULENT B.L.: $y_p^+ = (1 \sim 10)$ for models without wall function
    $y_p^+ > 30$ for models with wall function

NOTES:

- always check by visualizing the velocity vectors, that you have enough points in the boundary layer.

- velocity B.L. and thermal B.L. can be of different thickness in high-speed flows

- for inviscid calculations, there is no need to cluster points around walls. For computational efficiency, it is recommended that one uses different grids for Euler and Navier-Stokes simulations.

GRID CLUSTERING ELSEWHERE:

- locally refine grid for regions with large gradient, i.e. around shock waves, shear layers, slip lines, etc.

- shocks are typically resolved (smeared) through 3 cells in a spatially 2nd order accurate solver
    - in real life, shocks are paper thin (~0.2 mm)
    - a good CFD simulation has nice sharp shocks

- there might be need to iterate several times between grid generation – solution – flow visualisation, before an acceptable grid topology is achieved

NODE COUNT:

- a good guideline is to aim for about:
    - 40K ~ 60K cells in a 2D problem
    - 400K ~ 1 million cells in a 3D problem

- Large Eddy Simulation (LES) is typically in the order of 2~6 million cells for 2D

- Direct Numerical Simulation (DNS) is typically in the order of 10~12 million cells

- feel free to use very large cells far away the body (in the order of the characteristic length of the body), these cells are there to "give space" to the flow, and not for superior spatial resolution:

[n86]

Cell size at edges can be larger than the actual airfoil length.

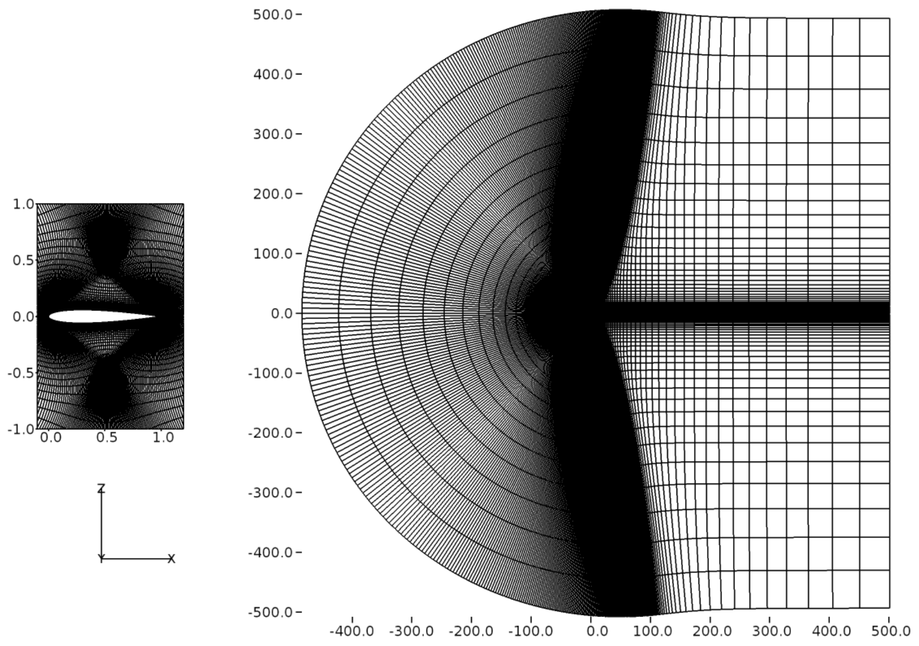


Fig. 8.35. "Giving space" to the flow can mean that cell sizes at the edges are much larger than the object examined.

GRID REFINEMENT STUDIES (for the "verification" steps):

- try to create 3 grid levels along the following lines:

MEDIUM grid:	- select grid density based on above guidelines
COARSE grid:	- remove every second gridline
	- means 4 x less nodes than medium in 2D
	- means 8 x less nodes than medium in 3D
FINE grid:	- halve each cell with a new gridline
	- means 4 x more cells than medium in 2D
	- means 8 x more cells than medium in 3D

- for unstructured meshes, halving or removing gridlines does not work, so aim for controlling 2 parameters at the same time:

  - element side length (try to halve or double this)
  - total cell (and not node!!!) count (aim for 4x more or 4x less in 2D, (8x more or 8x less in 3D)
  - you will likely get only approximately close to this

- for turbulent B.L.'s, watch for $y_p^+ > 30$ being maintained at all grid levels for turbulent models with wall function

- ***grid convergence*** means no change in the solution with increasing grid density, i.e. we look for MEDIUM and FINE grids yielding identical solutions

- ***optimum grid density*** means that the MEDIUM grid is not unnecessarily fine to capture the flow characteristics, i.e. that COARSE grid and MEDIUM grid solutions are different