Grid tips for CFD

M. Pourquie

April, 2010

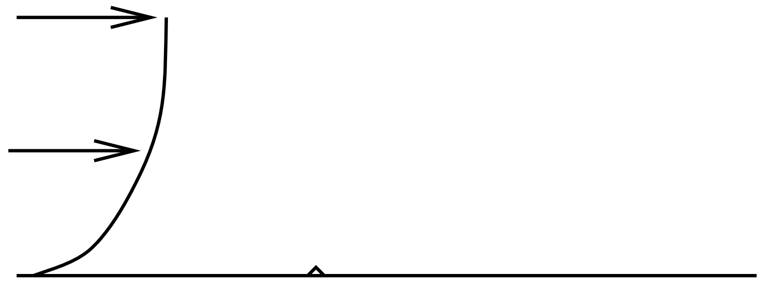
Contents

0.1 Introduction: about this document

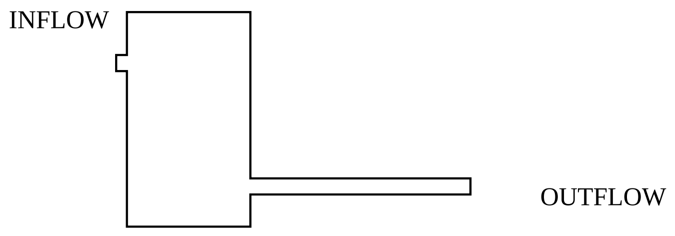
This document gives several tips for making more complicated geometries and more difficult grids. This is done by giving several examples. It will probably not give exactly the geometry you are interested in. Use both your common sense and these examples.
NOTE: for readability the default grid color in Fluent, yellow, has been changed to red.

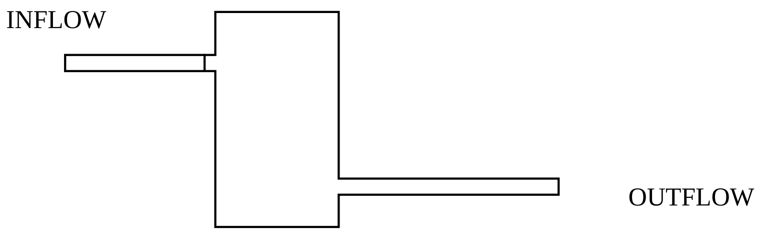
0.2 Geometry

- if possible, avoid unnecessary details
- see the figure, which has a small obstacle inside a turbulent boundary layer, and we are interested in the total heat transfer along the surface
- a small obstacle can of course sometimes be important, for instance it can trigger transition



- take into account the effect of inflow conditions
  - prescribing a constant velocity means that the flow not not fully developed
  - you may need an extra piece of inflow domain if the flow is to be fully developed.
  - check velocity and pressure to see if the flow is sufficiently developed after the inflow boundary

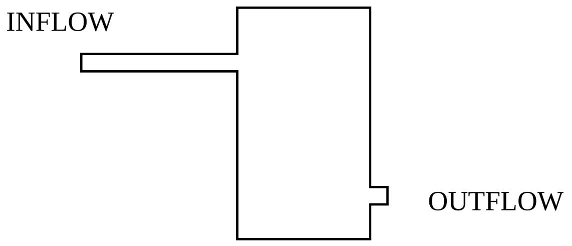


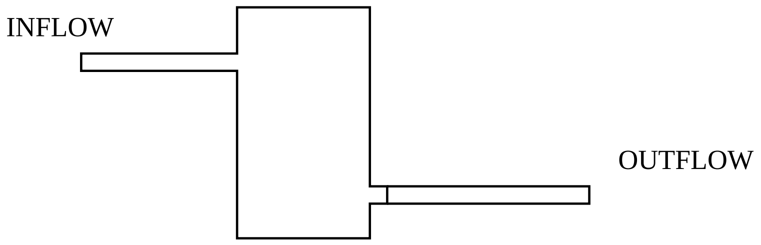


To put an extra piece of domain you could use sweep:

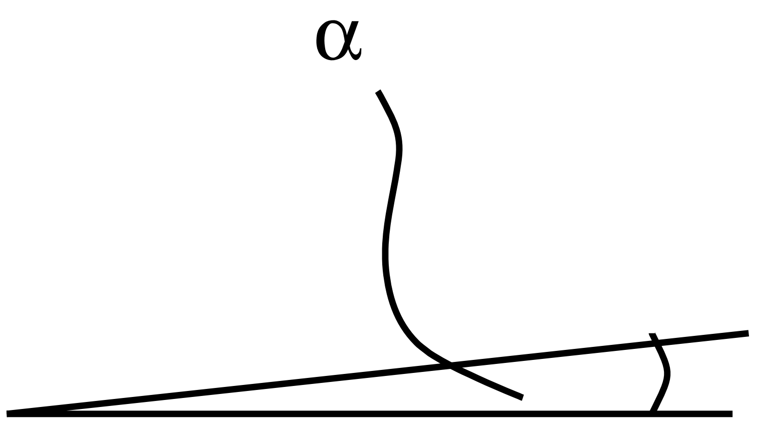
- geometry, 2D geometry, right-mouse-click create using wireframe, choose sweep, choose edge, define vector, sweep

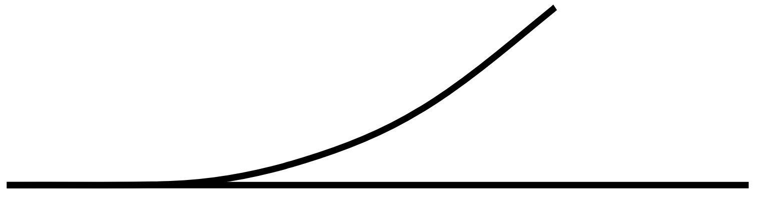
- take into account the effect of outflow conditions
  - Fluent outflow condition assumes fully developed flow
  - you may need an extra piece of outflow domain if the flow is to be fully developed
  - check velocity and pressure to see if the flow is sufficiently developed at the outflow boundary at outflow.





- avoid small angles and especially avoid cusps
- a cusp (see figure below) is a curved line touching another one, i.e. it intersects the other line while it is tangent too (for instance a round cylinder on a floor)





Cusp

0.3 The grid. Distribution of grid points.

The amount of grid points you can use is limited in practice. Limitations are set by the size of the memory and the amount of CPU you can use.

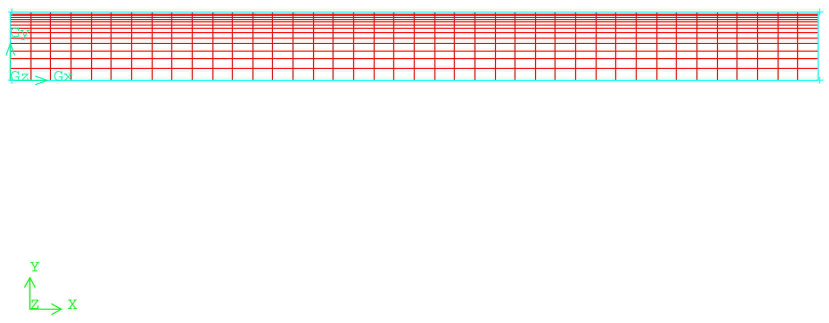
- not all regions of your domain need a very dense grid
- you only want a dense grid where it is important
- but when is it important to resolve things well?
- this can be a very difficult question in general
- and it depends on what quantities you are interested in

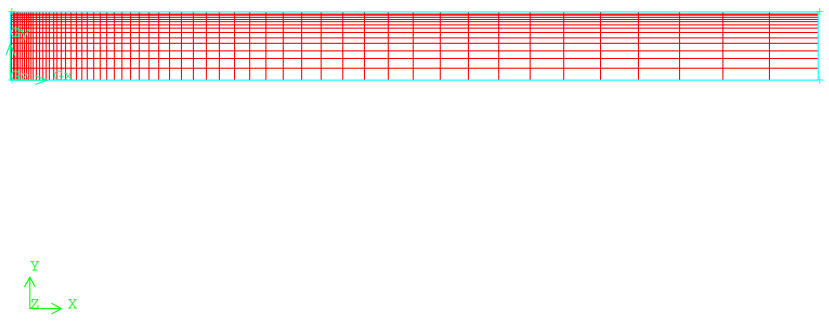
We now show some rules of thumb and examples, use experience and common sense.

Example: flow through a pipe.

- near a wall gradients in velocity change a lot
- if the inflow is a plug flow, the gradients in the velocity and the pressure change a lot at the inlet
- IF you are interested in what happens at the inlet you need a fine grid there
- IF you are only interested in obtaining a well developed flow this is not so important
- see the two examples below, the first grid could be useful for the case without interest in the inflow region, the second with interest in the inflow region.
- whether the grid is really good depends on the flow (Re number, type of inflow condition)

Grid for pipe geometry. Horizontal: x axis, Vertical: r-axis, r-axis from 0 to 0.5.



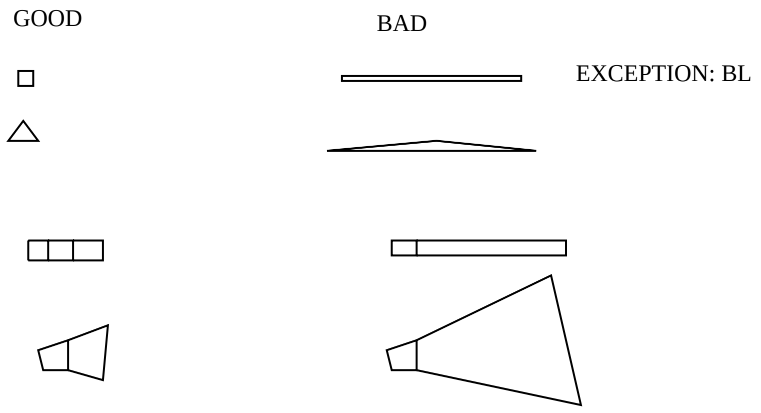


Example: flow around an obstacle.

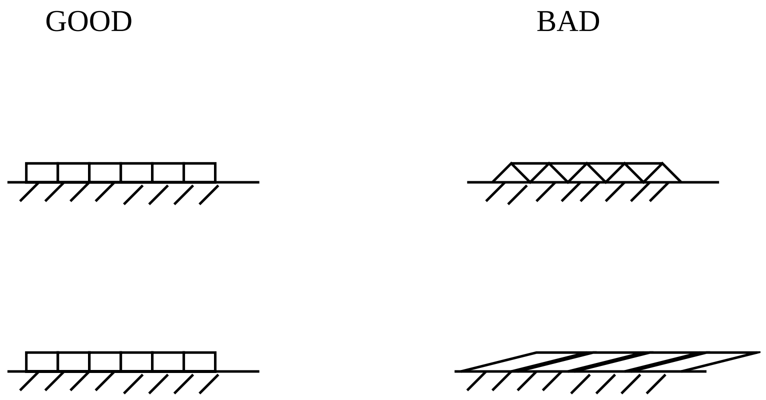
- near a wall gradients in velocity change a lot
- especially important if you are interested in wall stress or heat flux which depend on derivatives
- especially separation points are very sensitive
- the position of the separation point influences the wake region
- this, in turn, influences the total drag on the obstacle

0.4 Constructing the grid

- in general, good looking grids are good grids
- good looking means: no elements which are almost degenerate
- good looking means: transition of one element to the other must be gradual
- good looking quads (four-sided elements): as much like a square as possible
  - aspect ratio not more than 1:10
  - exception: boundary layer (BL) very close to wall
- if you need triangles: as much like triangles with equal sides as possible



- NO triangles at the wall
- try to have rectangles perpendicular to a wall

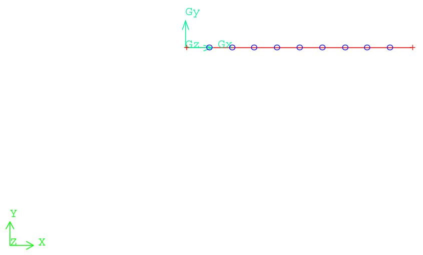


0.5 Building up a 2D grid: 1D grids to get 2D grids

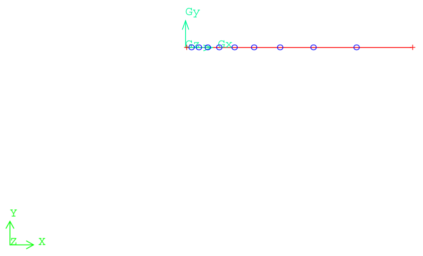
- a 2D grid is built up from the edges to the interior
- this holds for user-constructed grids and for automatically constructed grids
- so if you already have a 1D grid on the edges this controls the creation of the 2D grid
- in the following, I assume you can do 1D grids so try this first
- examples of 1D grids
  - uniform
  - size of number of elements
  - constant ratio
  - first length
  - last length
  - extra option: two sided
  - try out other options yourself
- create an edge
- 1D mesh in gambit: operation, geometry (yellow cube), select mesh edges
- select the edge, zoom in
- the edge has an orientation, a red arrow, which indicates which point is the first point, and which is the last
- the orientation can be relevant if you have a non-equidistant mesh
- the orientation of the edge is reversed by middle-mouse clicking it

0.5.1 1D grids to get 2D grids

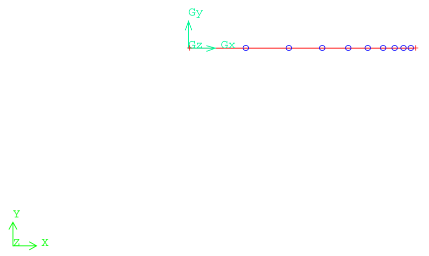
Uniform
Set internal size to an appropriate number (say, 1/10 of the length) and give enter
the mesh which will be generated is shown
click apply if you like it
alternative to internal size: interval count, select by clicking internal size, choose from drop-down menu
the second alternative is very useful when using non-equidistant grids, and when you produce block-structured grids, see below



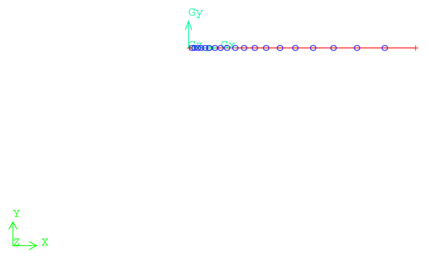
Non-equidistant meshes:
select the same edge, change Ratio to 1.3, apply



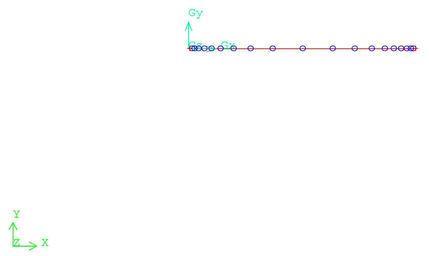
select the same edge, mouse-middle click (reverse orientation), change Ratio to 1.3, apply
The small grid cells are at the other side now



Alternatives, more control over grid cell size at start or end: change successive ratio to First length
choose First length to 0.01, interval count 20, apply. (Last Length is applied in same way.)



Choosing last length, double sided, length 1 and length 2 0.01, interval count 20
Refinement at two sides now



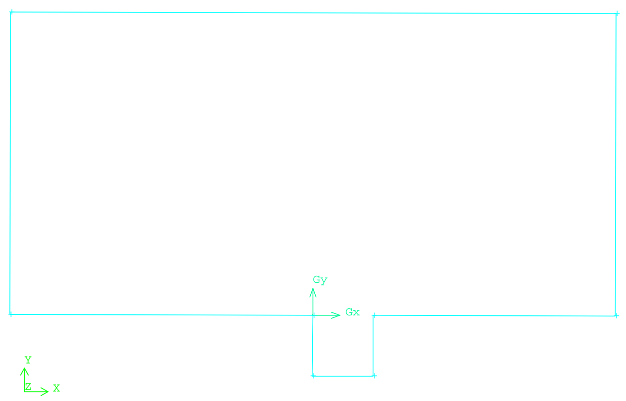
0.6 (Block)structured grids

- subdivide geometry in parts with four sides
- first make a mesh along the edges
  - opposite sides must have the same number of elements
  - use refinement near walls and corners
  - edges which have the same mesh can be meshed at the same time
- fluent should recognize the possibility of a structured mesh. It chooses quad, map. If not, do this manually.
- if gambit refuses to make the grid with quad, map: check if the number of grid points on opposite sides is always the same
- if the grid looks strangely skewed: check the stretch
- Advantages:
  - very good control over grid near wall or corner
  - often good grid quality
  - grid generation always successful (if block-structured is possible)

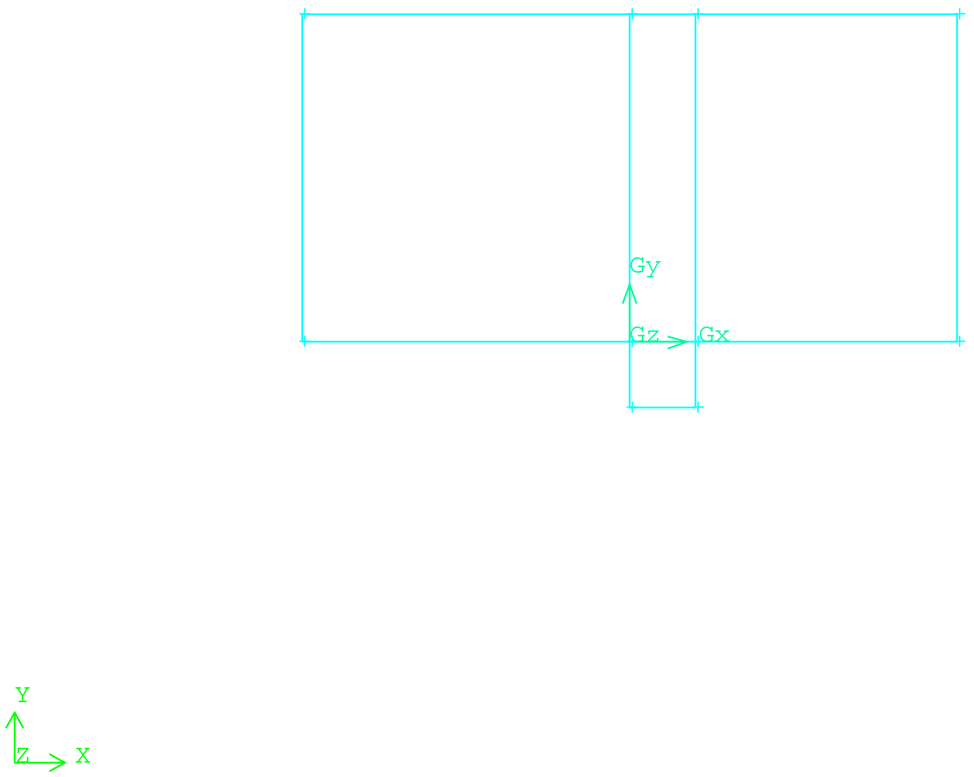
  Disadvantages:
  - more work for user, (create more surfaces)
  - possibly wasting grid points (possibly fine grid where it is not necessary, far away from the wall or obstacle)

For several geometries you need some tricks to get a block-structured grid.
A block-structured grid can be obtained if you sub-divide the geometry in faces
with each face having 4 sides.
I show some possibilities for a rectangular geometry with another rectangle sticking out
and rectangular grid with a round hole (2D cylinder) and polygons.

To get a structured grid we do not use this:

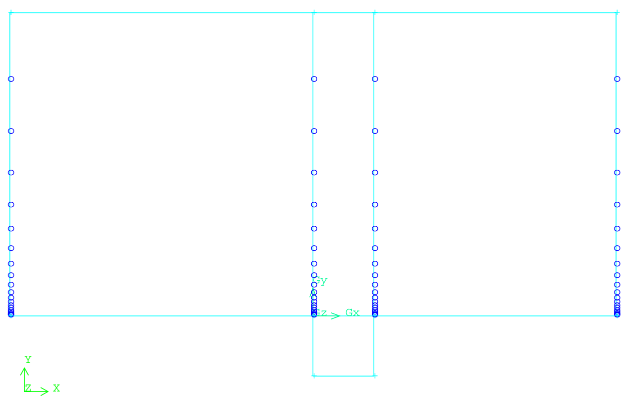


but this, subdividing into four faces:



Structured grids, gridding a cavity 1

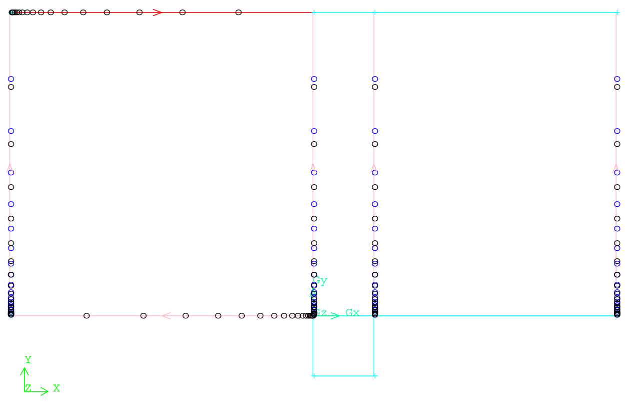
We start by making the 1D grids, you can do more than one 1D grid at a time

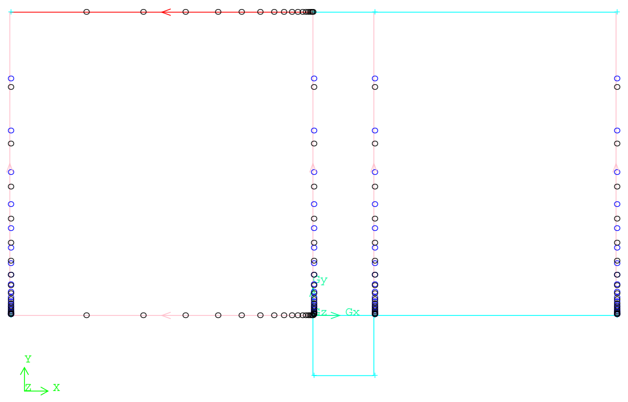


It may happen that the grid on an edge seems to have a "reversed" stretch

It has to do with the orientation (which side is first or last), see the red arrow

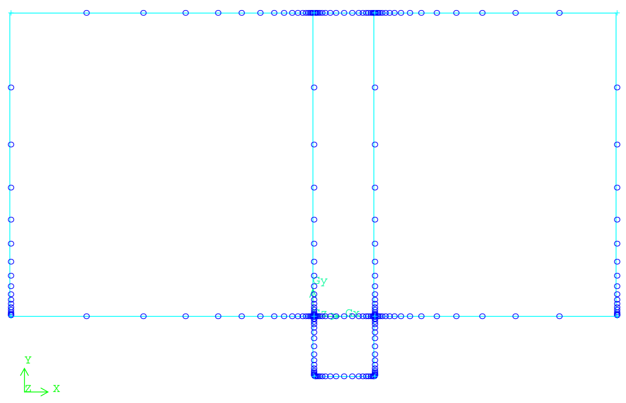
click the problem edge again with the middle mouse-button, the arrow/orientation reverses



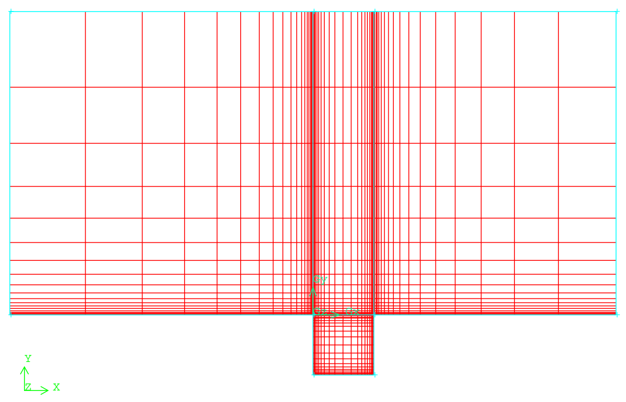


Structured grids, gridding a cavity 2

Finish the 1D grids (use "last length" and "double sided" for the sides of the cavity and corresponding edges, i.e. sides opposite to cavity walls, this gives small cells at both ends of the edge)



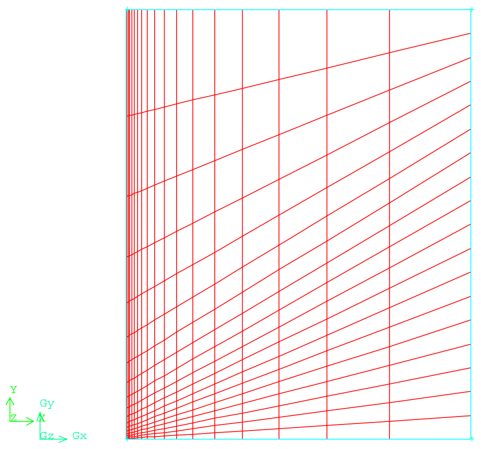
and generate the 2D grids, you can grab all faces at once and tell fluent to grid them



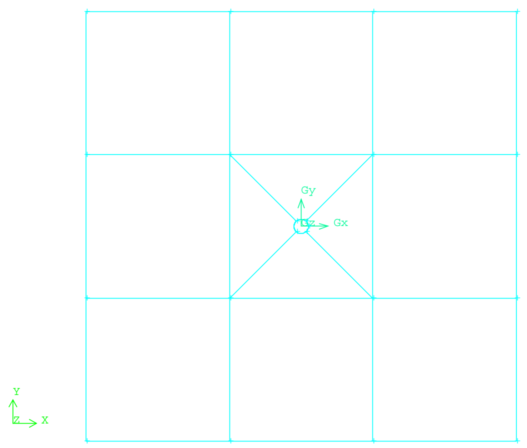
Structured grids, gridding a cavity 3

This is what happened with a structured grid with a stretched 1D grid on one edge and an equidistant 1D grid on the opposite edge

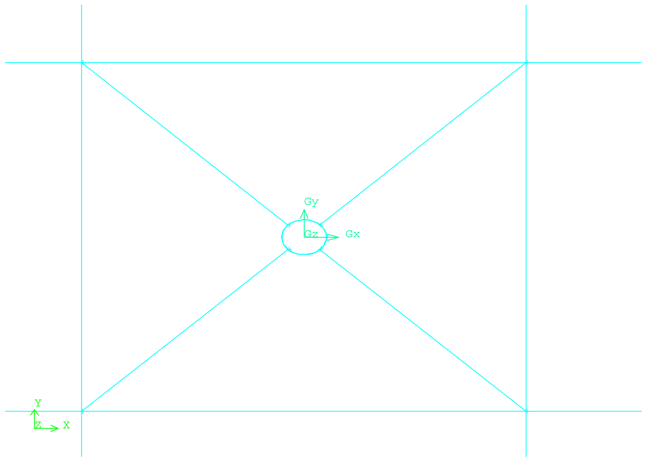
Remedy: re-grid the appropriate edge



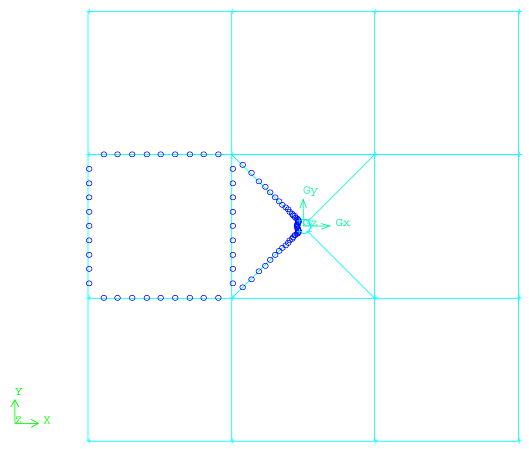
Structured grids, gridding a rectangle with a round hole 1
To connect the round hole with the rectangular blocks around using a structured grid subdivide the region in rectangular blocks, one block surrounds the hole and this block is subdivided in four regions



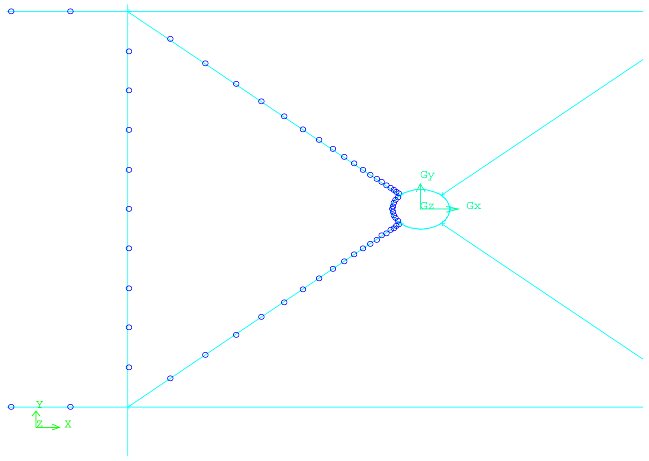
a zoom of the middle block with the hole, all faces now have four sides



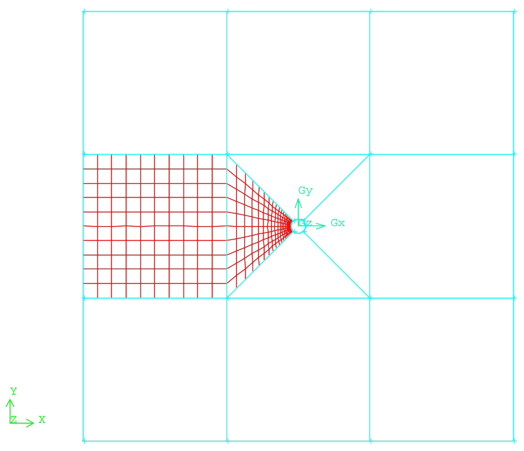
Structured grids, gridding a rectangle with a round hole 2
put a suitable 1D grid on the edges of the faces, I show it for two faces
in many cases we would also apply a stretch on the edges of the faces
outside the middle



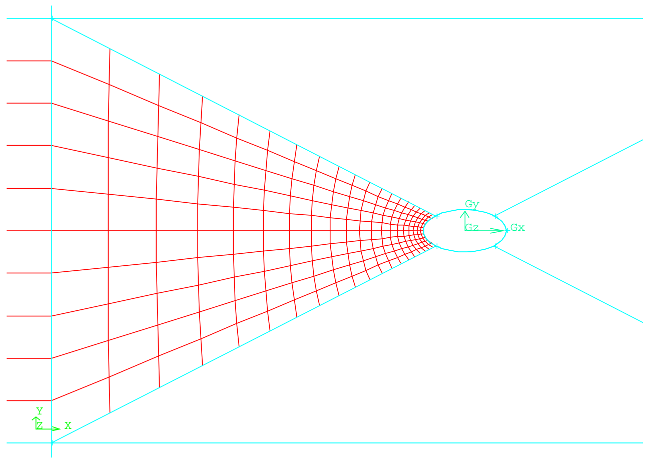
zooming in on the hole, note the stretch for some edges



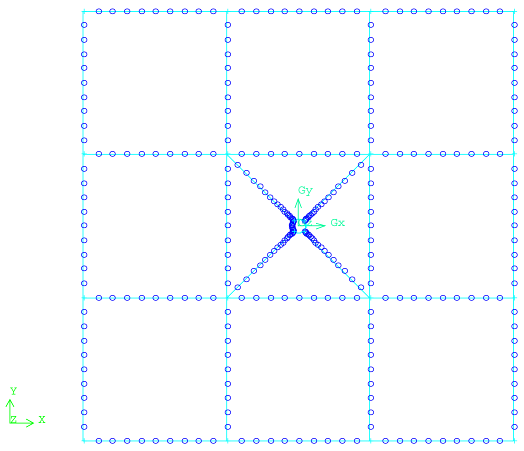
Structured grids, gridding a rectangle with a round hole 3
the resulting 2D grid

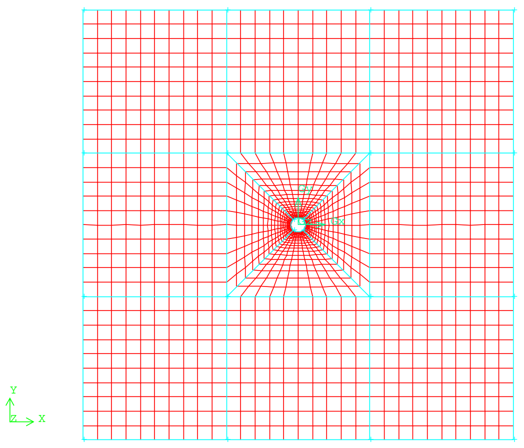


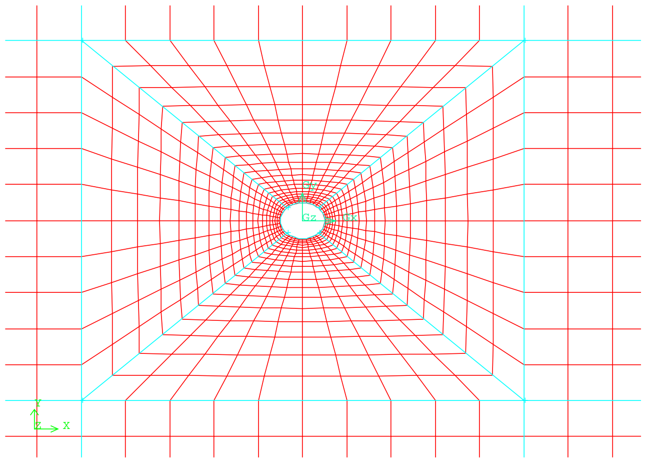
zooming in on 2D grid at hole



Structured grids, gridding a rectangle with a round hole 4
Complete 1D mesh, 2D mesh, with zoom on hole

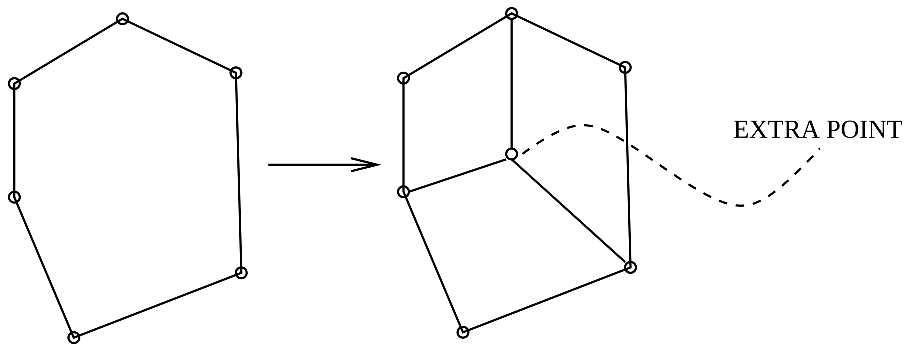


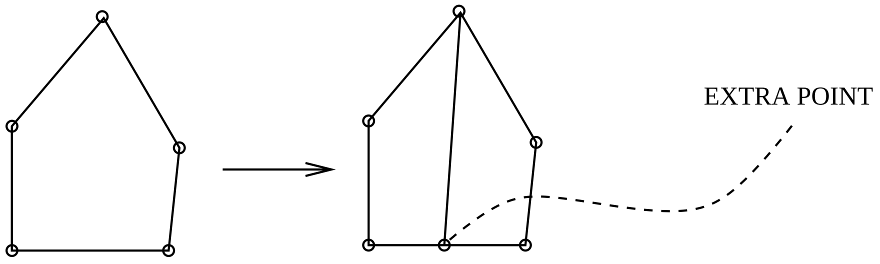




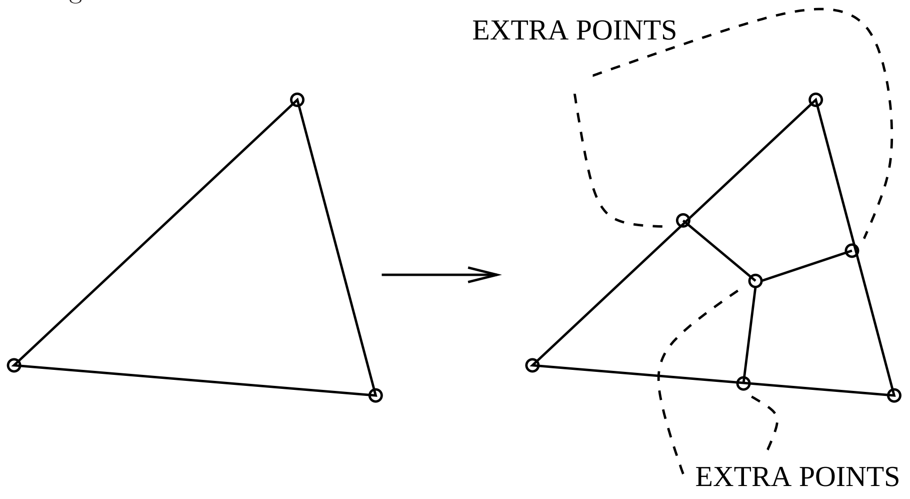
Polygons:

- I show tricks for polygons which are not too degenerate
- only the vertices and edges are given (no face yet)
- polygons with an even number of edges and vertices (say vertex 1, 2, ... 2n) can be subdivided by putting an extra point in the middle and forming quads by connecting this point with a vertex to vertex 1, 3, 5, ... 2n-1
- for polygons with an odd number of edges and vertices (say vertex 1, 2, ... 2n-1) subdivide one edge in 2 pieces and you get the problem for the polygon with an even number of vertices/edges, make sure that the 2 pieces are NOT in one quad





Triangles



0.7 Un-structured grids

Why un-structured grids, or why not
Advantages

- less work for user to create the geometry (no extra subdivisions in faces)
- should (almost always) work
- possible use: first impression of the flow

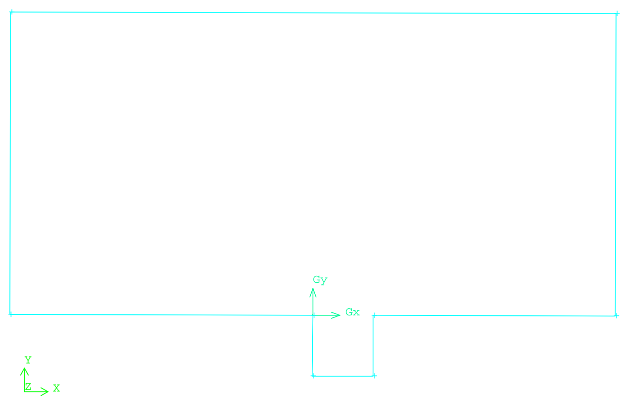
Disadvantages

- more work for gambit to create the grid
- unfortunately, it does not always work
- less control over region near wall, obstacle
- for too complicated geometry, grid generation may be only possible with triangles (triangles are less accurate and require more M and CPU than quads)

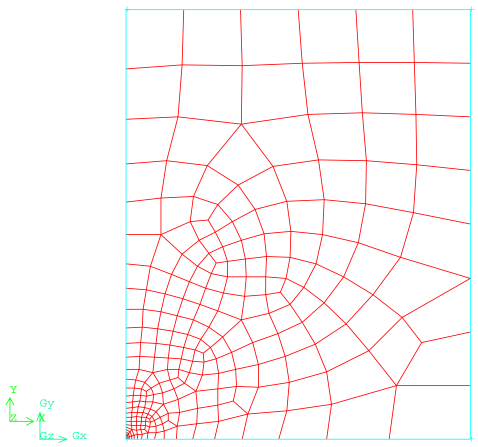
Notes

- user input may still be required (1D mesh, size functions)

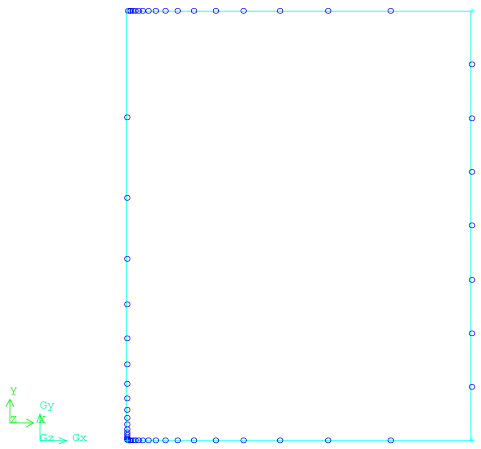
Example: The cavity



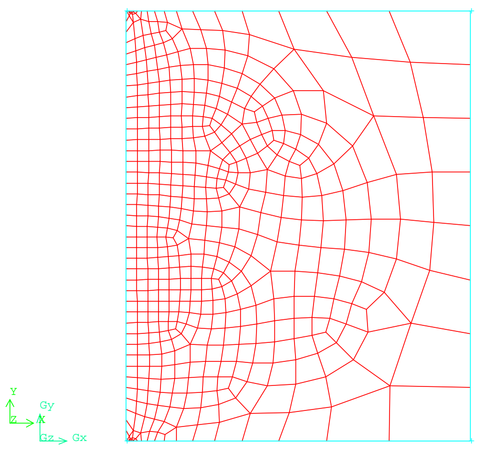
Some simple un-structured examples, using "quad, pave". The surface is a simple rectangle.
The difference is always the 1D grid on the edges
First one with grid which is fine near the lower-left corner, coarse elsewhere
This is accomplished with a stretched grid on bottom and left edge, uniform on top and right edge



Grid generation using pave does not always work. This one did not

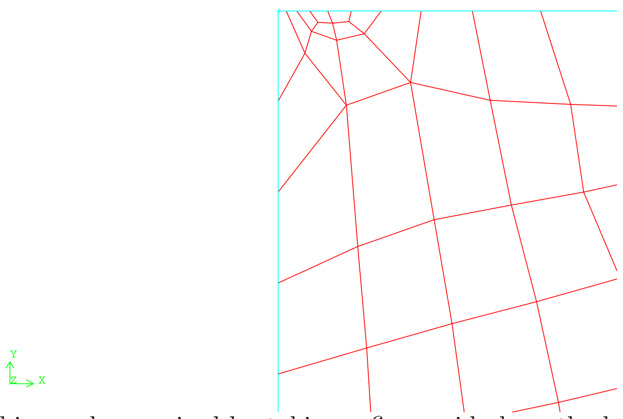


If the left side is part of an obstacle, we want a fine grid along this edge.
By varying the edge 1D grid we get the following
Take a fine grid along the left edge

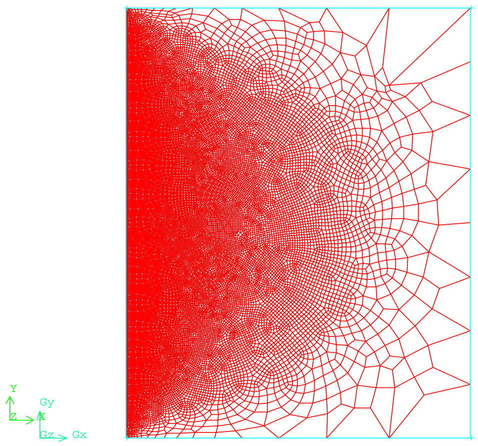


This gives an ugly upper-left corner, see zoom.

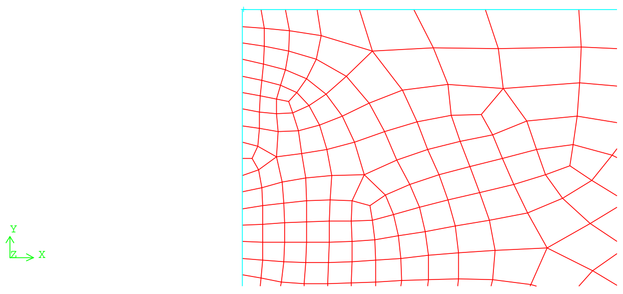
It gives a hint why the quad, pave grid failed for the same 1D edge grid as for the structured grid



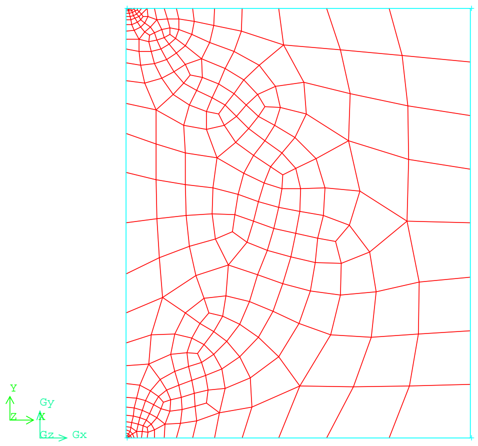
This can be repaired by taking a finer grid along the left edge (more M, more CPU!)



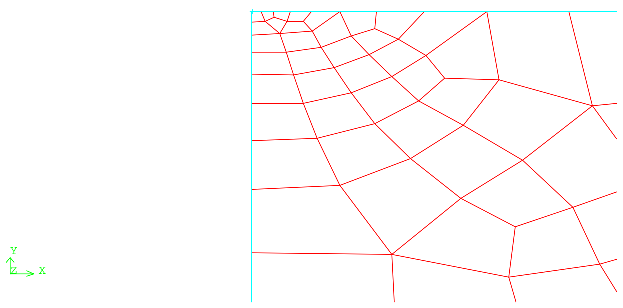
Look at the upper-left corner now.



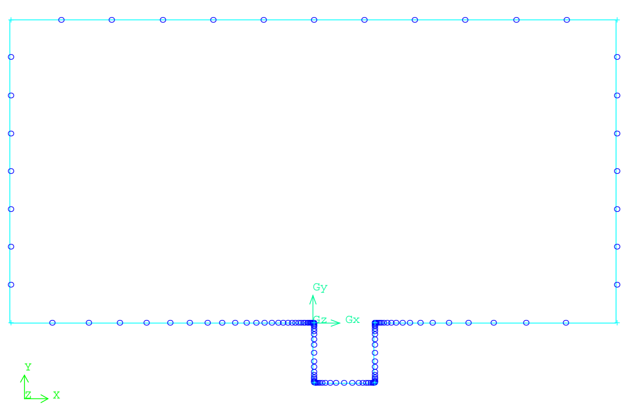
Taking a two-sided 1D mesh on the left gives quads near the middle of the left edge which may be too big in the left-right direction



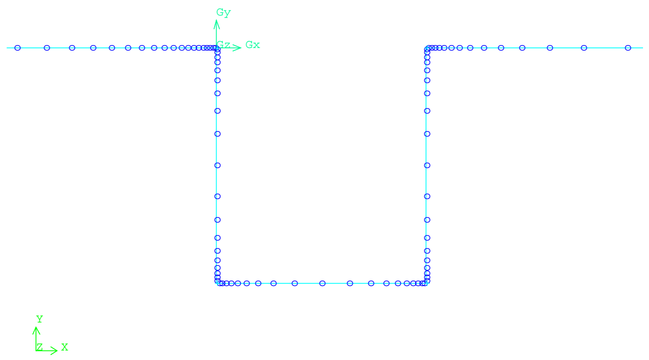
Also, the top edge is not really nice if it is a wall



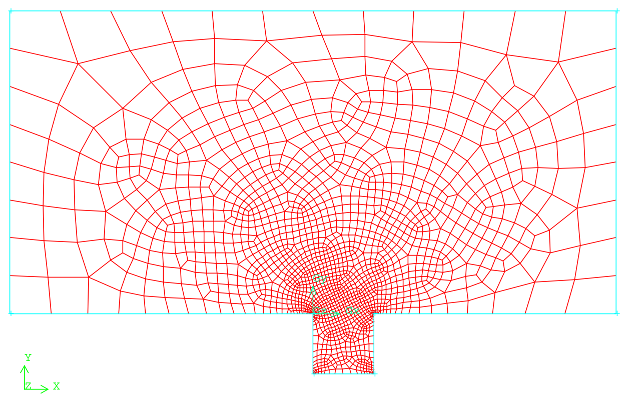
Doing the complete cavity can be done for instance starting with the following 1D edge grid



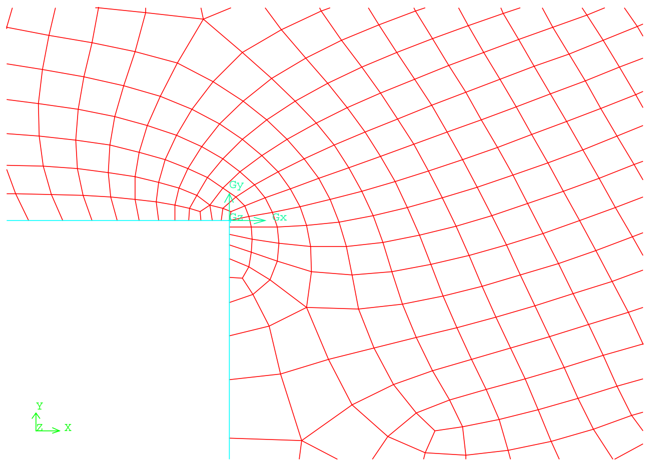
At the cavity I used two-sided refinement



This is the 2D grid



Zooming in the 2D pave grid at the corner of the cavity


Gy
Gx
Y
Z
X

Fluent offers a utility for making a high quality grid, (almost) orthogonal to the wall of an obstacle

It is called a Boundary Layer (BL) grid. This is a layer of rectangular cells which grows in a direction, normal to a wall

This works excellently in 2D for obstacles in free space, such as an airfoil

If the relevant geometry extends to the boundaries of the calculation domain it gives less beautiful grids

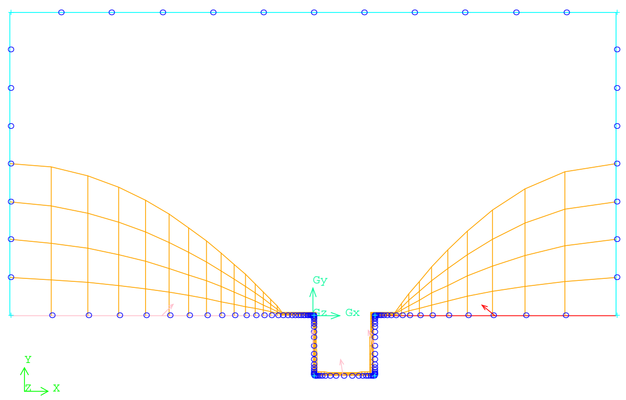
See the cavity flow, where we want to keep the wall closest grid point small along cavity wall and the horizontal walls before and after

A Boundary Layer grid

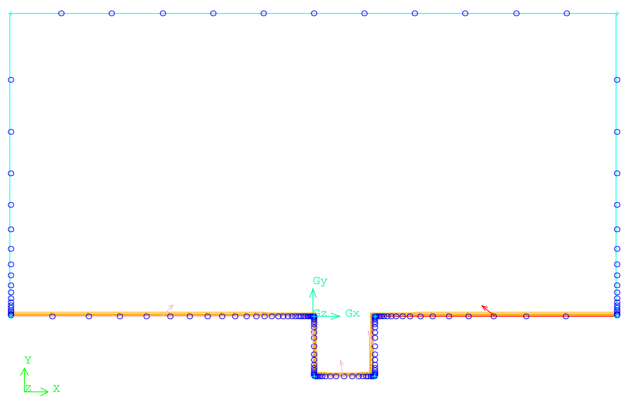
Gambit: grid, 2D grid, BL grid. Choose first row height, growth factor, number of rows. Choose edges. Use the sliders to change row height, growth factor, number of row to get a nice BL grid (see below).

First try: use the original 1D grid. Make a BL grid along the cavity walls and along the horizontal walls.

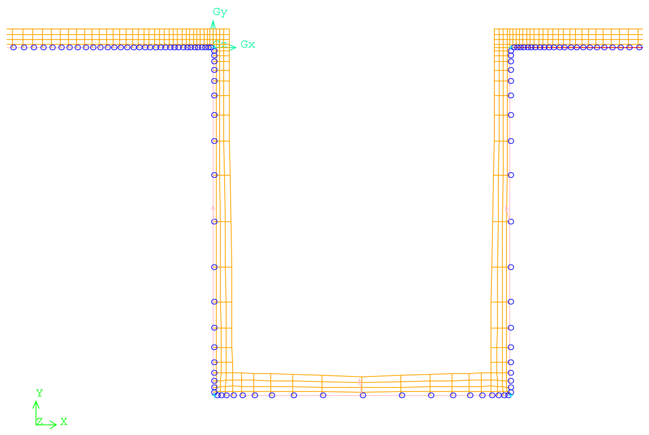
This gives a funny result, because the growth factors of BL grid and left and right edge of the geometry do not match.



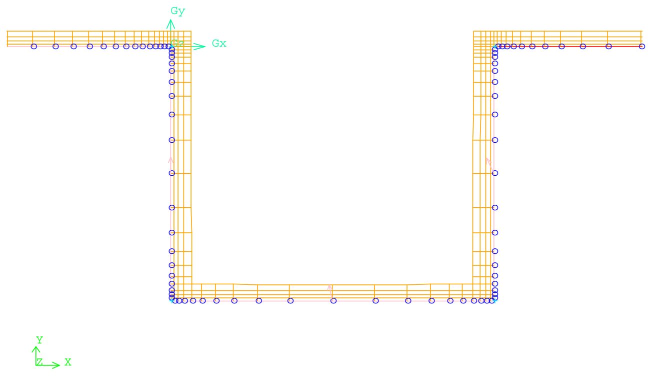
We match them by adapting the left and right edge 1D mesh



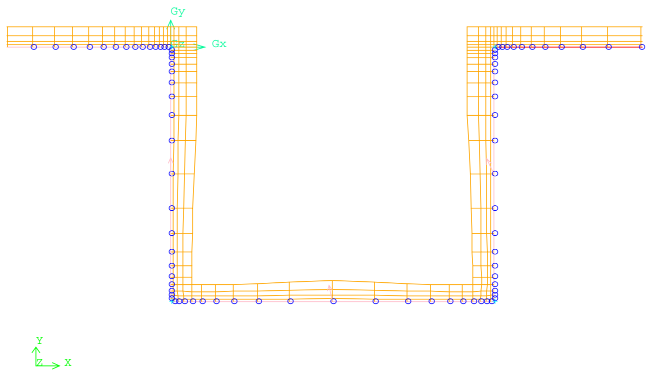
Zooming in on the cavity shows a little mismatch because the growth rate of BL and 1D wall meshes do not completely agree



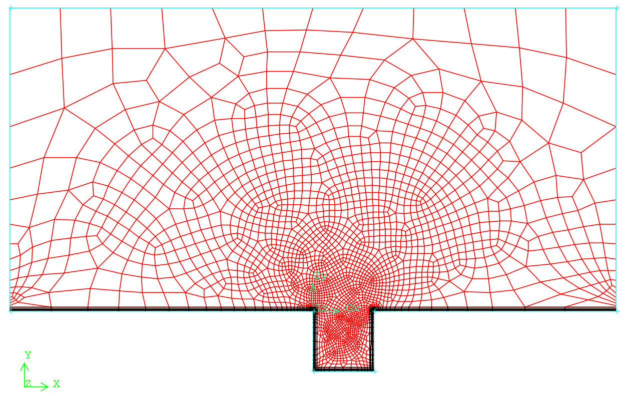
We can adapt these slightly, using the sliders, so that a more appealing BL grid results



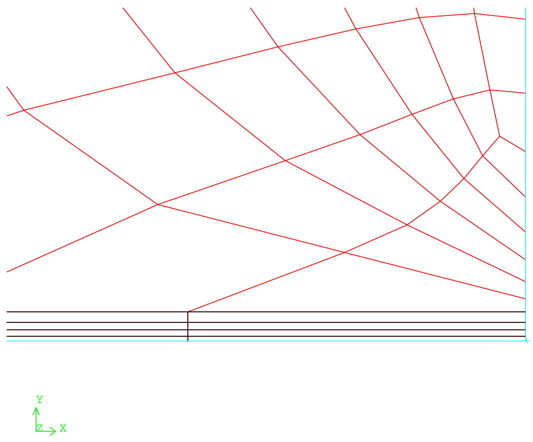
Changing it in the wrong way gives more mismatch (try around a little bit)



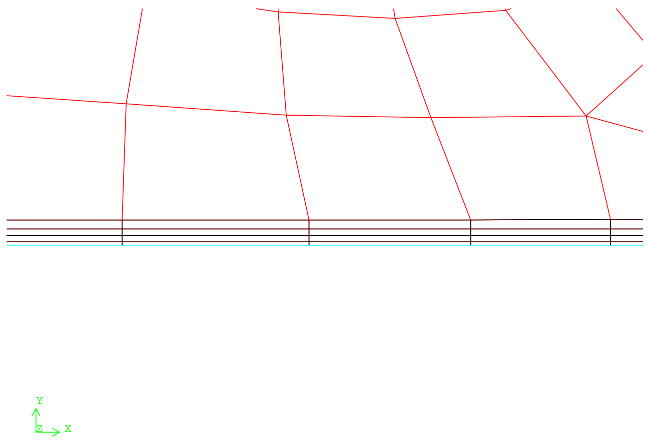
Zooming out, note the presence of flat BL cells near inlet and outlet
This may give (too) big square cells on top. The following 2D grid results



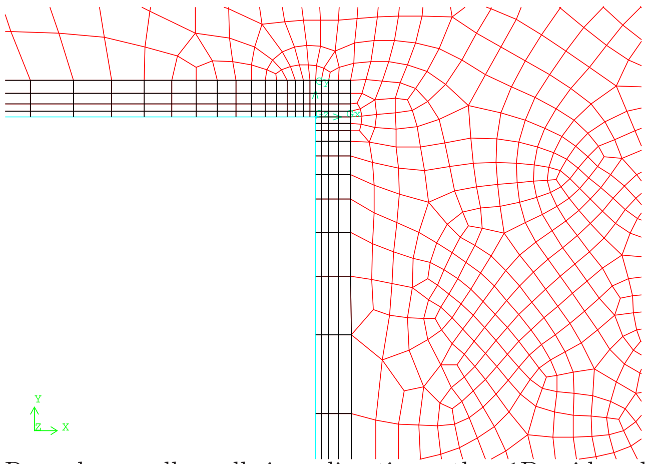
At the right and left end the grid does not look good.



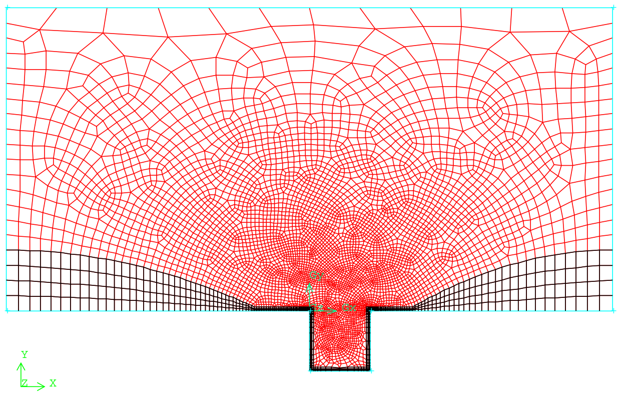
On top of the flat BL cells at the left and right horizontal walls we have big square cells on top of flat cells in the BL.



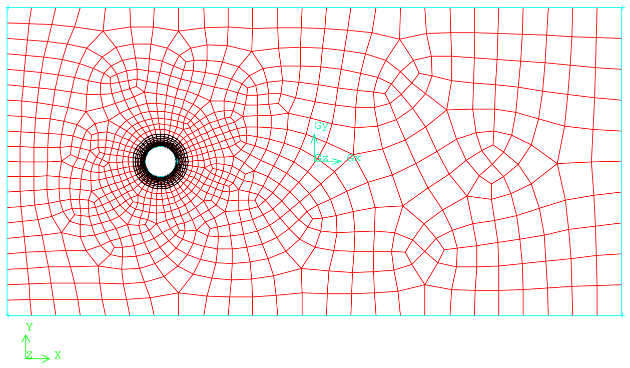
At and near the cavity things look good.



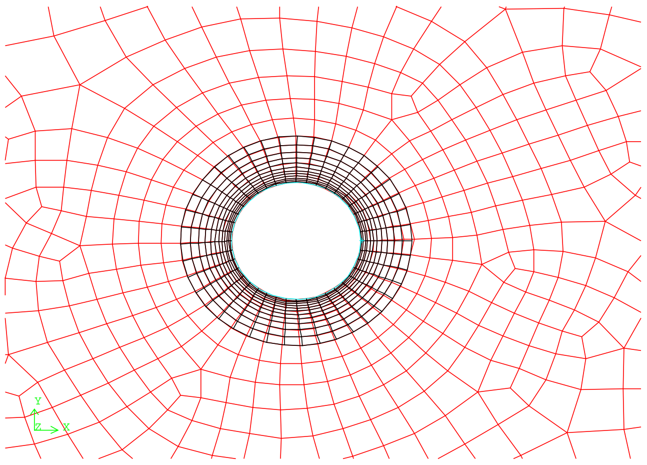
Remedy: smaller cells in x-direction, other 1D grid on left and right boundary. We use a "funny" BL grid together with these adaptions.



An obstacle far from the boundaries of the calculation domain does not have the difficulties encountered with the cavity.
See the following geometry, for flow around a round cylinder in a square region
No friction at the top and bottom horizontal walls, big cells allowed



zooming in on cylinder



0.8 Sizing functions (how to speed up/control unstructured grid construction)

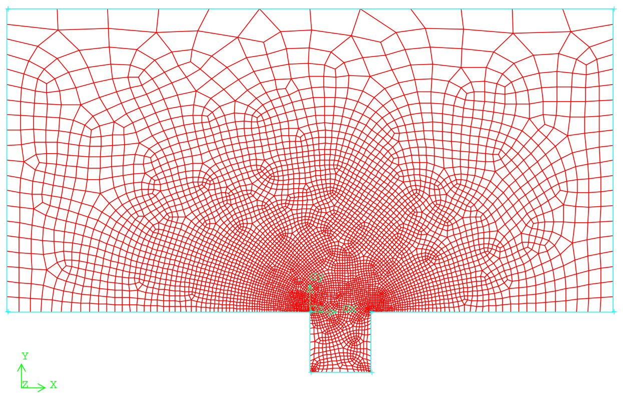
The way gambit makes a grid can be influenced by making a 1D grid.
Another possibility is using size functions
How a size function works:

- you choose a source, say and edge of a 2D region
- you choose an affected region, say a face of a 2D region
- you choose a growth rate (ratio size of neighboring elements)
- the grid size if Fixed (user specified in the sizing function window) or, alternatively, you can choose Meshed, than the size function takes the meshing from the source as a start. Then it lets the elements grow at the specified rate into the affected region.
- it gives you control how fast elements grow, and in which region

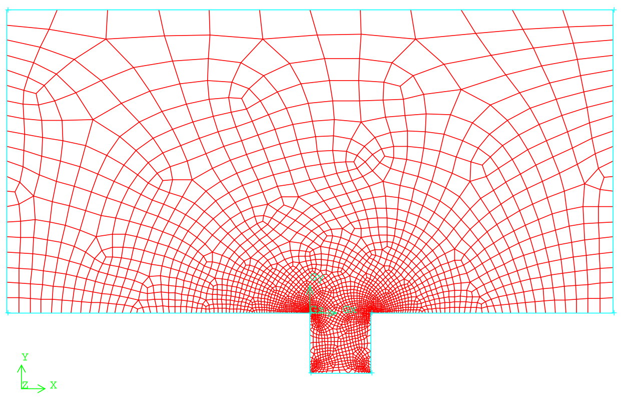
In gambit:

- tools,
- you choose a source, say an edge of a 2D region
- you choose an affected region, say a face (with this edge) of a 2D region
- you choose a growth rate (ratio size of neighboring elements)
- the size function takes the meshing from the source, and lets the elements grow at the specified rate into the affected region
- it gives you control on how fast elements grow, and in which region

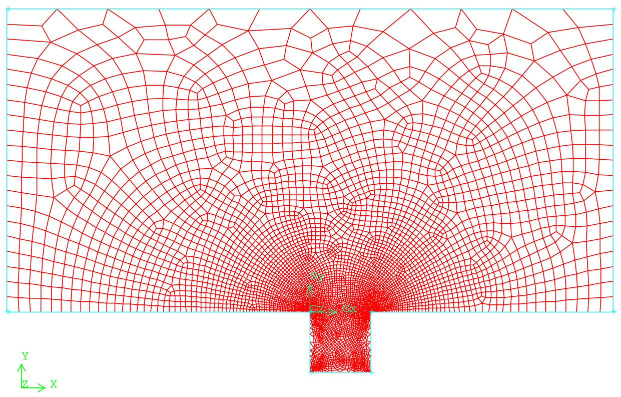
No size function, 1D grid on edges



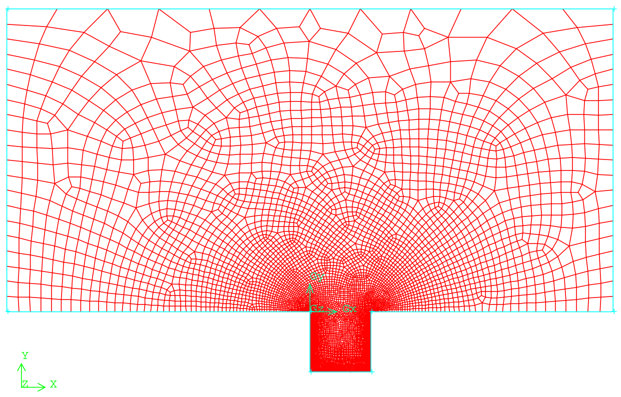
Sizing function and 1D grid, growth rate 1.1



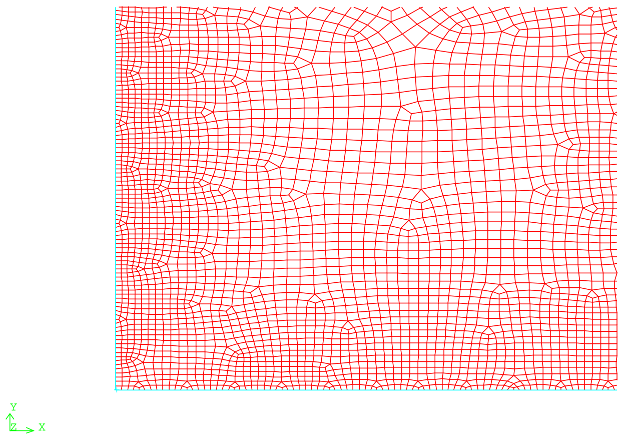
Sizing function and 1D grid, growth rate 1.05



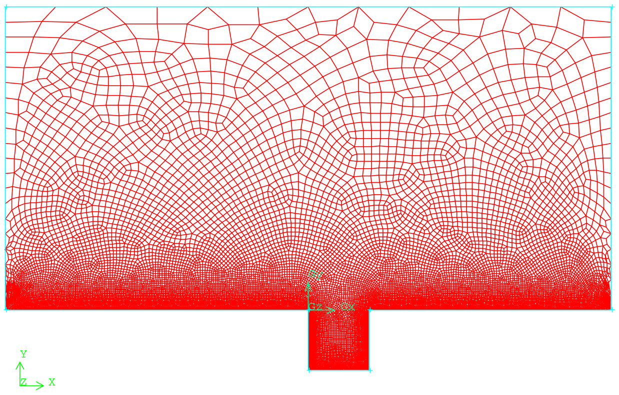
Sizing function and 1D grid, uniform 1D grid at cavity, growth rate 1.05



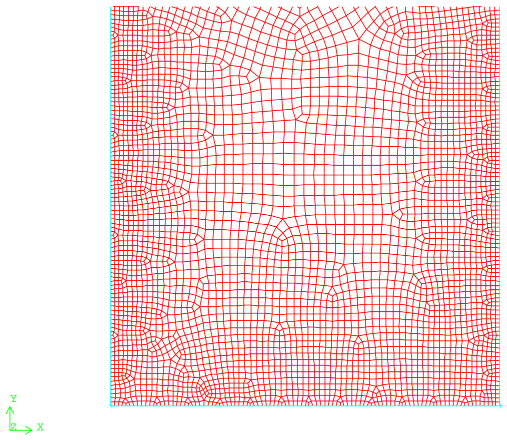
Sizing function and 1D grid, uniform 1D grid at cavity, growth rate 1.05, zoom on cavity



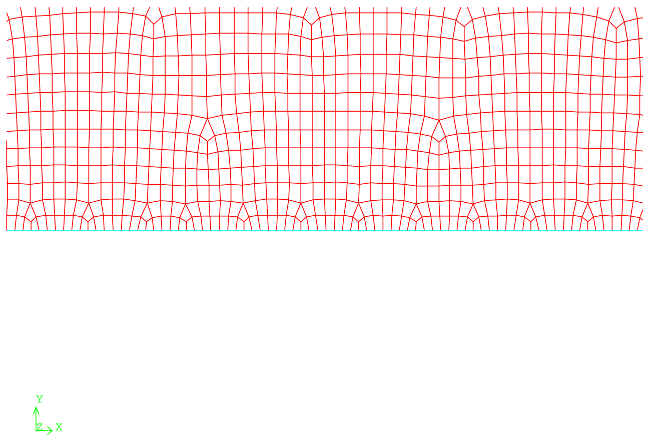
NO 1D grid, growth rate 1.05



NO 1D grid, growth rate 1.05, zoom on cavity



NO 1D grid, growth rate 1.05, zoom on horizontal wall



0.9 Avoiding difficulties: using unmatched grids

Fluent can couple two geometries that were generated independently, with grids that are totally different along the edge where they meet.
Example: a 2D channel with a very fine grid at the inlet, coupled to a grid which is coarser
Recipe for gambit/fluent:

- generate the two parts independently, and create the two mesh files
- the boundaries where they meet must overlap (at least after an additional translation/rotation)
- the boundaries where they meet must have BC interface in gambit (give them an easy-to-remember name)
- preferably, where the parts meet there should be no sharp (change of) gradient in the flow variables
- when you are done you can exit gambit

after generating the mesh files (say, mesh1.msh and mesh2.msh) we use a fluent utility

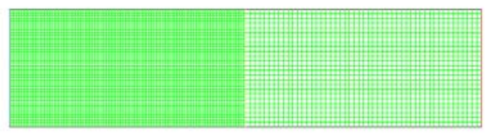
- (linux) type utility tmerge -2d
- you get a prompt, for each mesh file to be merged you have to give the mesh filename plus scaling factors in x, y directions (1 1 if none), translation (0 0 if none) and a rotation angle (0 if none)
- when you are done (enter instead of a grid name will stop the program), a name for the merged grid is asked (meshtot.msh, say)

Now in fluent

- read in the mesh file meshtot.msh, do the usual grid checks etc
- Main menu: define-grid interfaces, choose surfaces of interfaces, give the interface a name
- do NOT choose periodic or coupled (periodic ONLY for periodic cases, coupled ONLY for solid-fluid)
- create
- proceed as usual in fluent

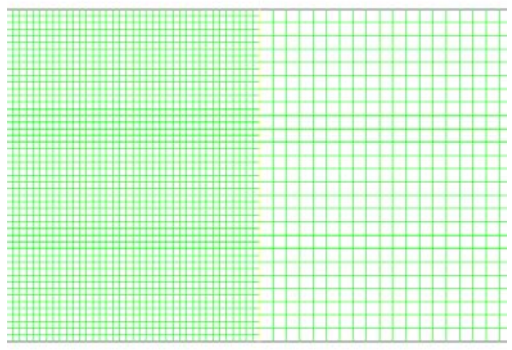
Results on the next page

A non-matched grid with refinement factor 2 in both directions (the grids do NOT have to be uniform)



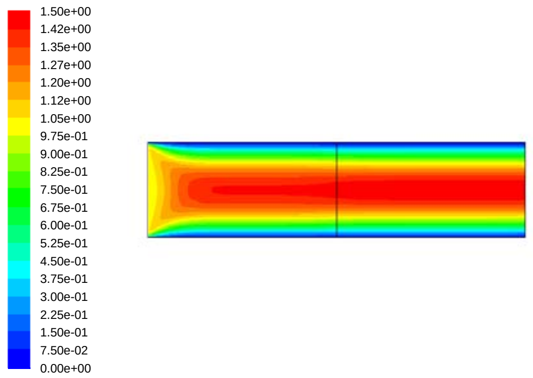
Grid   Jun 19, 2007
FLUENT 6.3 (2d, dp, pbns, lam)

Zoom of the grid



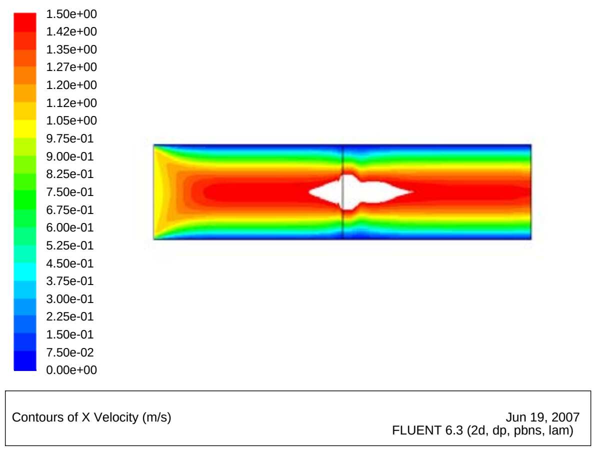
Grid   Jun 19, 2007
FLUENT 6.3 (2d, dp, pbns, lam)

The u velocity profile, there is a slight change as we cross the interface



Contours of X Velocity (m/s)   Jun 19, 2007
FLUENT 6.3 (2d, dp, pbns, lam)

If you refine a factor of 5 the program does not like it, see the jump in U near the interface (we use the same contour values as above, part is outside the contour range)



0.10 Grid-refinement tips (save work when you do a calculation of a coarse and a fine grid)

Fluent can interpolate a solution from one grid to the other. Interpolation from a general grid to another is very inaccurate, it is zero-th order. It parses the points on the new grid and finds the point on the old one which is closest to this point. The variables in the new point are assigned the values in this closest point. Still this can be much better than re-starting.

Moreover, many variables and settings, like material properties, inflow velocities, and the position of user-defined curves for plotting, can be written to a file which can be read in by another fluent run. This avoids lots of re-typing of numbers and is especially also useful for grid refinement studies.

Steps for grid refinement (we use the text mode, some parts are also possible in click mode but perhaps not all):

- step 1: make the same geometry with a coarse grid and a with a fine grid save both cases, export a mesh file for both cases
- step 2: generate a solution on the coarse grid, this solution does not have to be a solution which is converged up to machine precision but it has to look physical (no strange big velocities, strange big vortices etc)
- step 3: write the boundary conditions etc to the boundary conditions file, this will save the boundary conditions but also save material properties and for instance user defined plotting curves

  file, write-bc
  give a filename and enter
- step 4: we write the coarse grid solution to a s-called interpolation file which can be read in later on when read the fine grid

  file, interpolate, write-data
  give a filename
  the package asks: all fields?
  answer is usually: yes (unless one of the fields is useless)
- step 5: write-case-data for the coarse solution (you never know!)
- step 6: read-case, read the fine grid
- step 7: /file/read-bc to read the boundary condition file
- step 8: /file/interpolate/read-data
  give the filename
- step 9: check (contour plots etc) if the interpolated solution looks as expected
- step 10: continue the solution on the finer grid

Fluent can also add grid points, so it can adapt the grid. It can do so using a criterion like the magnitude of a variable, its gradient, or the curvature (the change of the gradient). An example follows.

BEFORE you use adapt for adapting the grid: the grid will be changed and CANNOT be changed back, do file- save/case-data first if you think you want to keep the old stuff

from the main menu:

adapt-gradient

default is curvature (=change of gradient)

choose a variable which you want to use for refining the solution (can be pressure, velocity, all kinds of turbulent quantities)

compute the maximum and minimum:

compute

choose a number above which you want refinement (a value lower than Max)
mark

the output window tells how many cells were marked

manage-display

shows the marked cells

if they are OK with you (otherwise change the criterion and try again)

adapt

in the Gradient Adaption window, fluent asks:

yes

have a look at the adapted grid

display/grid

continue the calculation, fluent interpolates automatically to the new grid

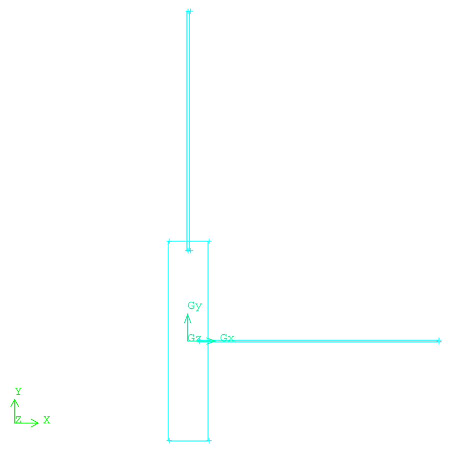
Other criteria for refinement besides gradient can be:

iso-value, region, volume, yplus....

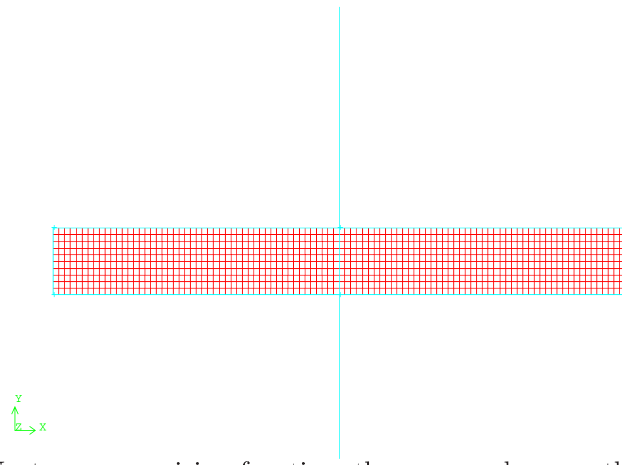
Using the new solution, you may want to refine again. Do not forget save case-data if you do!

0.11 Size functions for small parts sticking into big parts

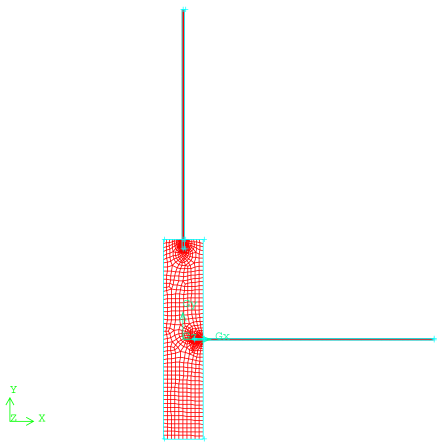
A geometry consists of a big central part with inflow, outflow part which are much smaller sticking out



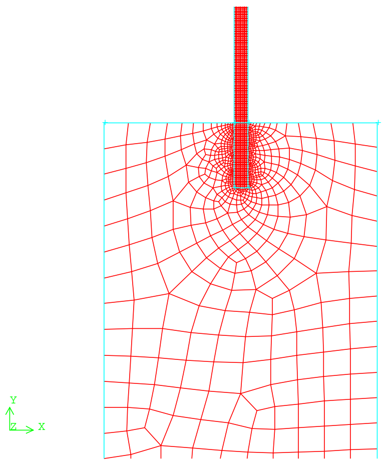
We use size functions. Make the side parts stick INTO the big part. Split the geometry so that we have 5 separate parts which are connected (use connect curves to make sure they are connected). Grid the side parts which are sticking in and the parts which are sticking out. An example for the horizontal side part, the region near the big vessel is shown.



Next we use a sizing function, the source edges are those of the side parts which stick into the big part. The target face is the face consisting of the central parts minus the side parts sticking in. (Size function: click the "toolbox" icon to the left-top. A new row of icons pops up. Then click size function, the yellow icon which is the second icon from the right in the row that pops up after clicking the toolbox. We used growth factor 1.3 and maximum cell size 0.05 (the diameter of the big part is 1). We also took "gridded" instead of "fixed" so the grid of the side parts will be used. After definition of the size function grid the 2D domain. The result



Zooming in:

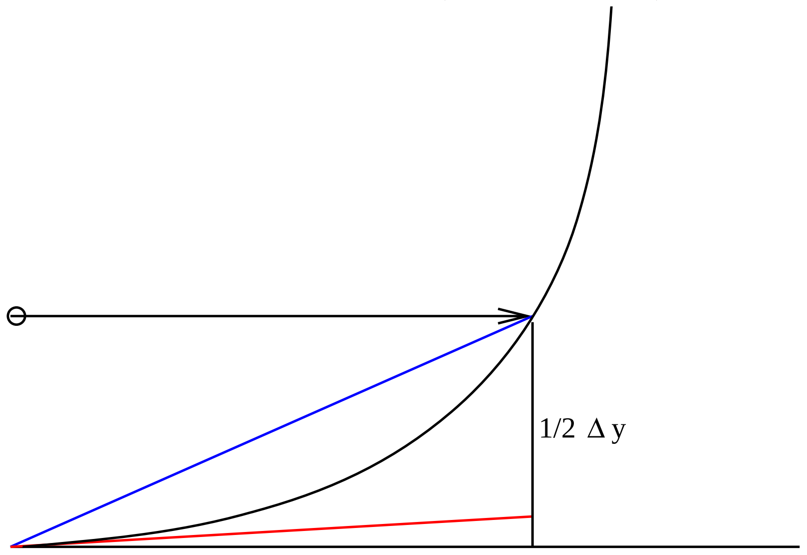


0.12 Grids for turbulent flow

- for a turbulent flow, the wall region is rather special
- at the wall, so-called wall modeling is used

A turbulent flow has a very high change of gradients close to the wall. This means that we need many, very small grid cells near the wall. Very small ones just at the wall, and we have to go to bigger cells very gradually.

See the following figure which shows what happens if the cells at the wall are too large. Remember that the wall shear stress is calculated with $\mu \frac{\partial u}{\partial y}$. We approximate $\frac{\partial u}{\partial y}$ with $\frac{\Delta u}{1/2 \Delta y}$ to estimate the derivative. This amounts to approximating the velocity curve with a straight line (linearisation). The tangent of the straight line and the curved velocity profile near the wall, the derivative is clearly under-estimated (and the stress too).



From the figure we see that on a coarse grid the wall shear stress if not calculated correctly.

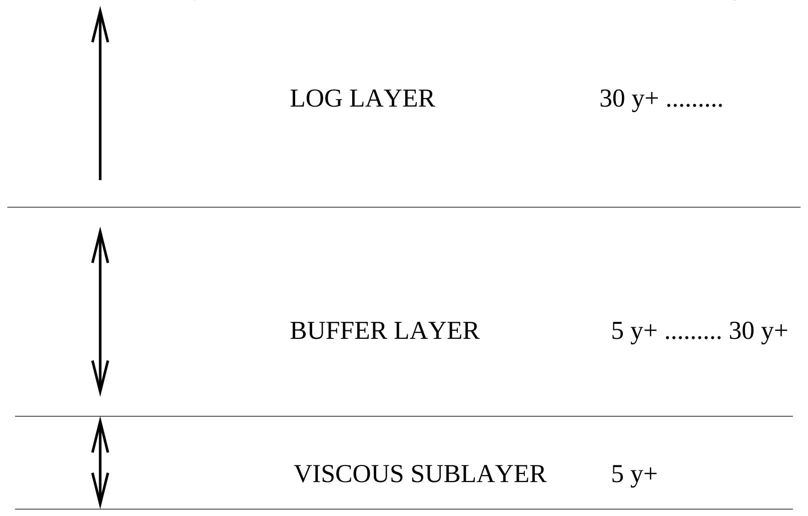
Note that the problem is in the tangential velocity component. This is the velocity component related to the wall shear stress. Because of the grid requirements, the wall region is often not explicitly calculated but it is modeled. The modeling applies to the tangential velocity component. What the model does, is to estimate the wall shear stress. The model (called a wall law) is derived using turbulence theory for the flow near the wall of a boundary layer or channel. It is can only be used if the first gridpoint is at a suitable distance from the wall, this distance should not be too big but also not too small!

- the CFD package calculates a dimensionless distance from the wall, called y+
- check (plot) this distance when you finish a simulation
- this distance y+ should be EITHER
  - $y+ < 2.$ or less, use 'enhanced wall treatment'
  - OR $y+ > 20$, use 'standard wall law'
- if you have large parts of the near-wall grid with $5 < y+ < 20$ (the buffer region) use enhanced wall treatment but the solution will be less accurate

If your grid cells are in the buffer region, you can either make the grid cells smaller in wall-normal direction OR make the grid cells bigger in the wall normal direction!

The use of a wall law has some perhaps unexpected effects on the plots you make. Look at the velocity profile near the wall for a simulation with $y+ < 5$ and for a simulation with $y+ > 20$. The simulation with $y+ > 20$ will look 'kinky' near the wall when plotted because the curved velocity profile is modelled, not simulated, and the plot function will just connect points with straight line segments. It is one of the few occasions that a kink in the velocity near the wall should be there.

A short explanation is in place here. Take into account that we have a high Re number flow. Inertia effects are dominant over viscous effects, except when we are close enough to the wall. If you are close enough to the wall, viscous effects dominate. For this type of flow we have three regions, the viscous sublayer ($y+ < 5$), the log layer ($y+ > 20$) and in between the buffer layer. As the name says, in the viscous sublayer viscous effects dominate. In the log layer, inertial effects dominate. In the layer in between, both are important. See the figure below.



When you are deep enough in the viscous sublayer, a no-slip condition is used. The turbulence model is adapted in the wall near region.

When you are in the log region, there is a relation between the tangential velocity and the wall stress which can be used.

When you are in between, an empirical averaging of no-slip and wall law stress is applied.

The wall treatments are developed really only for standard boundary layer or channel flows. If there is separation or if there is an adverse pressure gradient, it does not hold exactly. These regions should always be considered with caution.

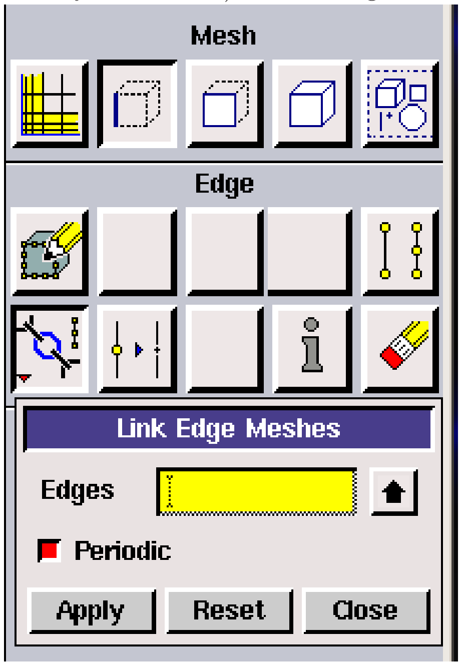
In Fluent, automatic grid refinement is possible with as criterion 'wall yplus'. The automatic grid refinement procedure has been described above.

Turbulent inflow conditions.

- the default turbulent inflow conditions $k = 1$ and $\epsilon = 1$ are usually nonsense
- do not use k and ϵ, but use turbulence level and length scale (click the black arrow to the right of the "k and ϵ" button on the inflow window)
- set turbulence level and length scale to reasonable values
- for a channel: 3% of turbulence and $\frac{1}{10}$ to $\frac{1}{20}$ of the channel height
- if you have no idea what to do else, use "compute from inlet" to initialise the turbulent quantities

0.13 Periodic boundary conditions

First you have to link the mesh on the sides with periodic meshes. In gambit, under mesh, choose 1D grids (edge command button), under Edge right-mouse click the left icon from the second row of icons (see figure). Choose the two sides which have to be periodic, apply. Do this BEFORE generating a 1D grid on these edges. make sure the orientation (the arrow shown on an edge when you select it) is in the right direction, if not, use shit-middle-click to reverse the direction.



Now, under boundary conditions, you can choose both edges in one periodic boundary condition.

0.14 Time-dependent simulations

In fluent, define-models-solver. Choose between first- or second order. Under solve-iterate set a physically meaningful time-step, and a number of time-steps. You can start up the unsteady solution from a steady solution.

The error (residual) shows a saw-tooth like behaviour, because a time-dependent problem has to be solver every time-step, i.e. the error starts high at the beginning of the time-step, and becomes smaller as more iterations are done. You can have too few iterations per time-step (no convergence) or too many (residual is no longer changing). Adapt number of iterations per time-step or time-step (why this last choice?).

0.15 Conclusion

- grid generation is not plug-and-play
- changes in one region affect other regions as well
- you learn it by doing it
- you have to be practical, sometimes you have to accept some waste of CPU or a region with a less quality grid
- remember: good-looking grids are good grids