051176 - Computational Techniques for Thermochemical Propulsion

Master of Science in Aeronautical Engineering

Mesh generation in OpenFOAM

Prof. **Federico Piscaglia**

Dept. of Aerospace Science and Technology (DAER)
Politecnico di Milano - Italy

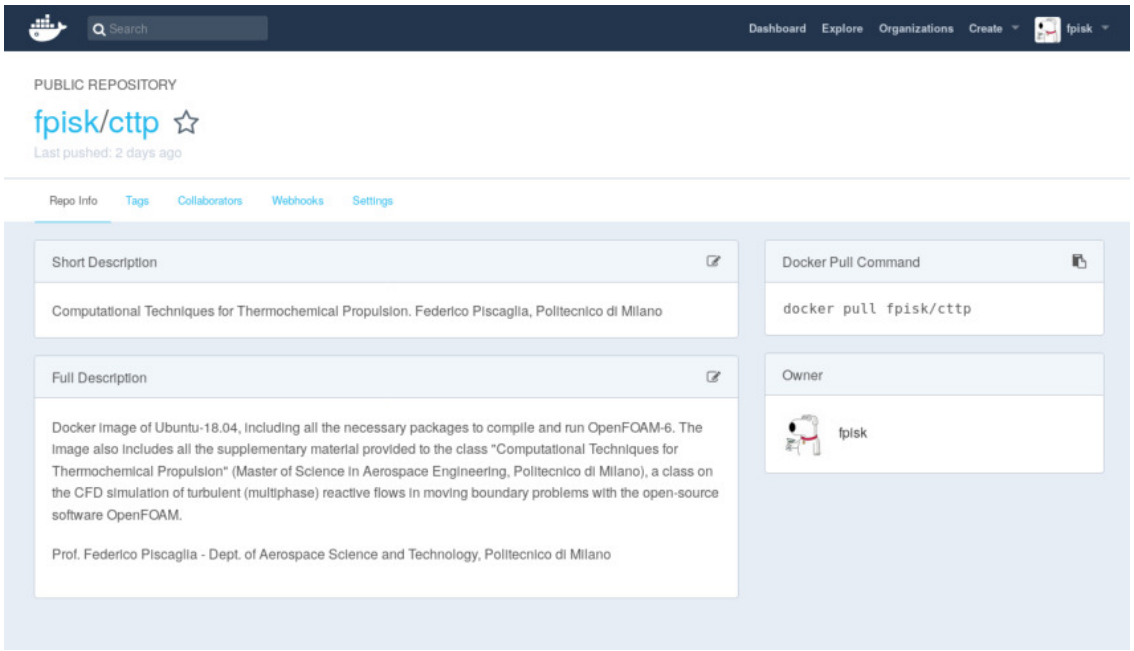
federico.piscaglia@polimi.it

Bibliography

https://cfd.direct/openfoam/user-guide/v6-snappyhexmesh/

Bibliography

Docker image for the class → `fpisk/cttp`

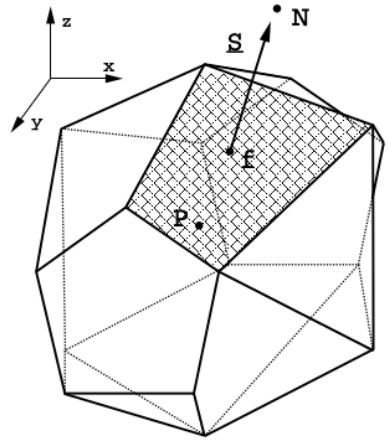


You can reproduce the steps of this presentation by OpenFOAM-7.

Table of contents

Mesh description in OpenFOAM

By default OpenFOAM defines a mesh of arbitrary polyhedral cells in 3-D, bounded by arbitrary polygonal faces, i.e. the cells can have an unlimited number of faces where, for each face, there is no limit on the number of edges nor any restriction on its alignment.



A mesh with this general structure is known as a `polyMesh`; this type of mesh offers great freedom in mesh generation and manipulation in particular when the geometry of the domain is complex or changes over time.

Mesh description

The `constant` directory contains the description of a finite volume mesh in a subdirectory `polyMesh`, whose definition is based around faces:

- internal faces connect 2 cells and
- boundary faces address a cell and a boundary patch.

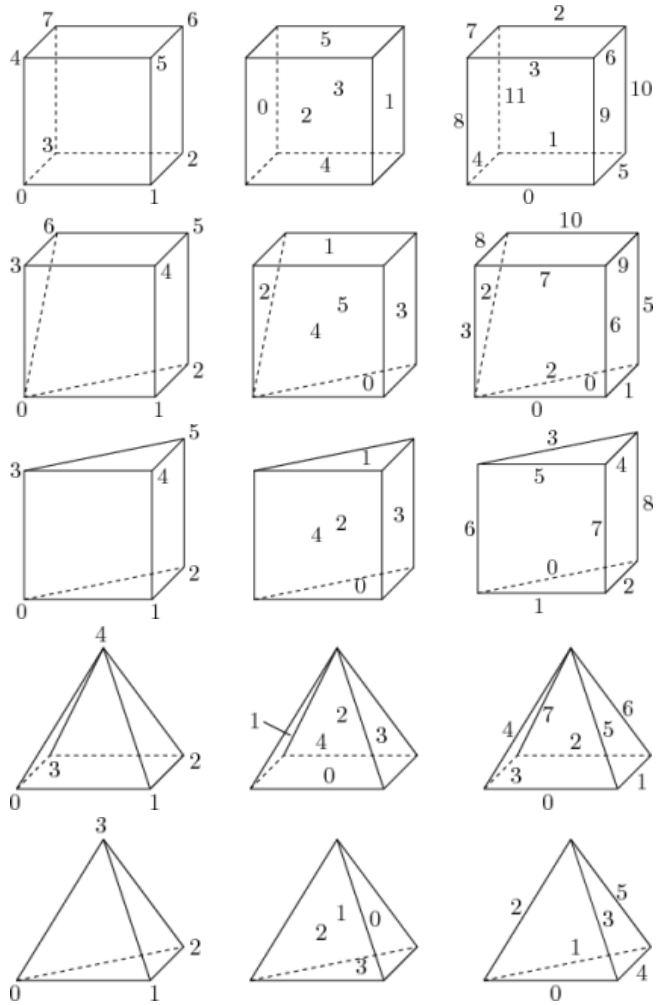
Each face is therefore assigned an 'owner' cell and 'neighbour' cell so that the connectivity across a given face can simply be described by the owner and neighbour cell labels. In the case of boundaries, the connected cell is the owner and the neighbour is assigned the label '-1'. With this in mind, the I/O specification consists of the following files:

Cell shapes

In addition to topological data, the finite volume mesh incorporates information about its geometric entities:

- volume of CVs,
- the area of faces
- centroids of elements and faces
- the alignment of faces with the vectors joining the centroids of the owner and neighbor elements, etc.

Depending on the `cell shape`, the geometrical features of the cell entities are calculated. In OpenFOAM, cells are treated as polyhedra.



Mesh description

OpenFOAM is designed as a code for 3-dimensional space and defines all meshes as such.

- However, 1- and 2- dimensional and axi-symmetric problems can be simulated in OpenFOAM by generating a mesh in 3 dimensions and applying special boundary conditions on any patch in the plane(s) normal to the direction(s) of interest.
- More specifically, 1- and 2- dimensional problems use the empty patch type and axi-symmetric problems use the wedge type.

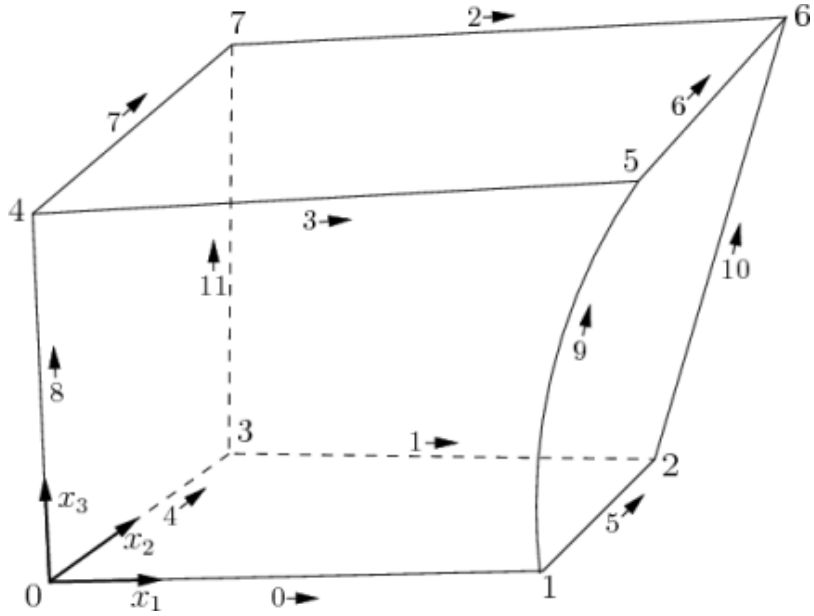
Generating a mesh in OpenFOAM

Two different automatic mesh generators are available in OpenFOAM®:

- the `blockMesh` utility for generating simple meshes of blocks of hexahedral cells;
- the `snappyHexMesh` utility for generating complex meshes of hexahedral and split- hexahedral cells automatically from triangulated surface geometries in STL or Obj format.

blockMesh

Mesh generation with the `blockMesh` utility



source: https://cfd.direct/openfoam/user-guide/v6-blockmesh/

snappyHexMesh

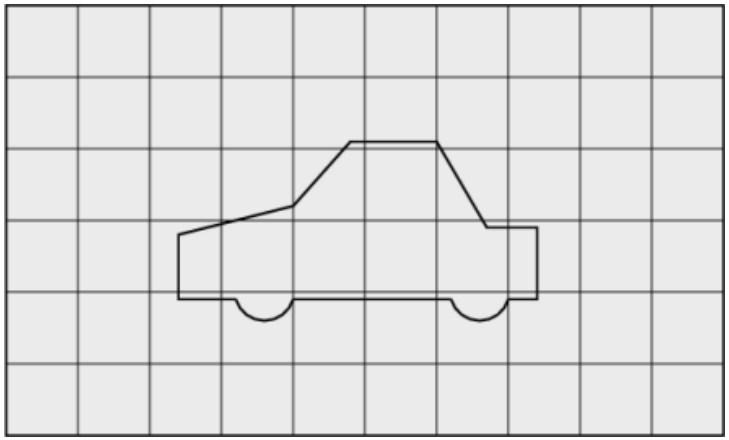
Prerequisites to run `snappyHexMesh`

- a **CAD software** to edit/repair the input geometry; `snappyHexMesh` works with CAD files in stereolitography format (STL), that are also used for 3D printing. Free software to manipulate STL files are freeCAD or MESHMIXER by Autodesk.

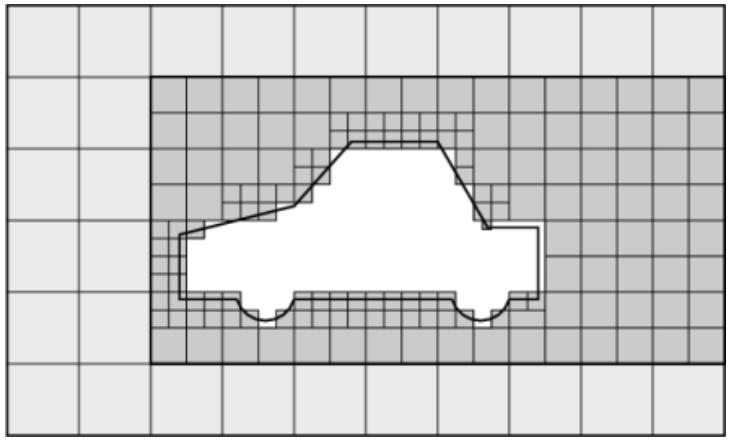
- **System requirements:**
  - OpenFOAM installed (in this presentation version 7 is used);
  - a good amount of memory: `snappyHexMesh` makes use a significant amount of RAM memory during its operation (about 4 GB of RAM per million of cell volumes generated). If you want to generate large grids, consider to use dedicated resources (computational nodes);
  - Message Passing Interface (`MPI`) installed on the machine for parallel computation;
  - `ptscotch`: software package and libraries for sequential and parallel graph partitioning, static mapping and clustering, sequential mesh and hypergraph partitioning, and sequential and parallel sparse matrix block ordering;

Mesh generation with `snappyHexMesh`

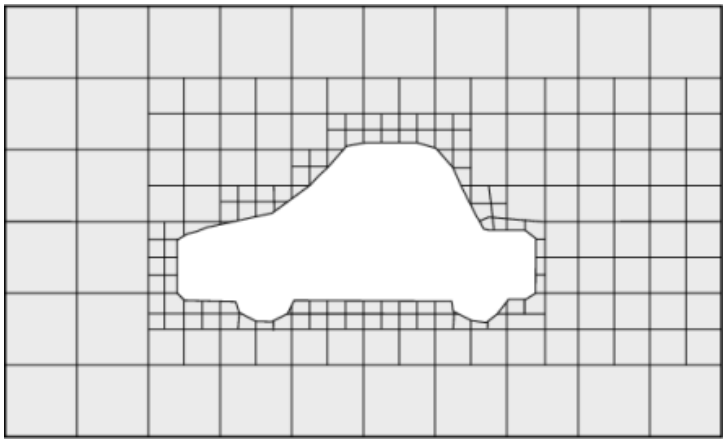
1) Background mesh



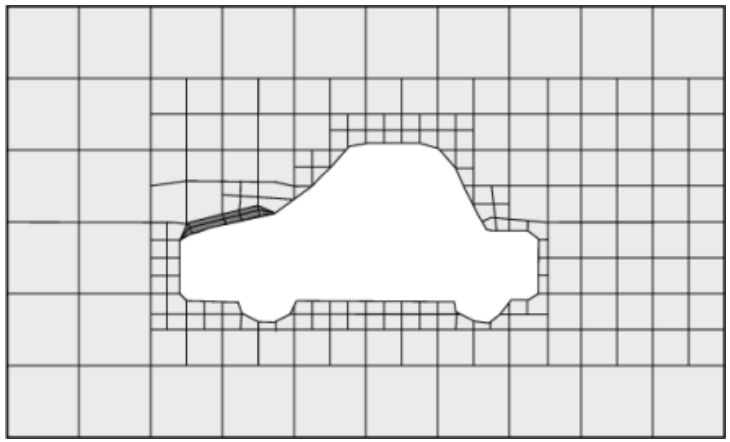
2) Castellated mesh



3) Snapped mesh

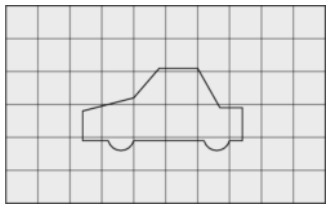


4) Layered mesh

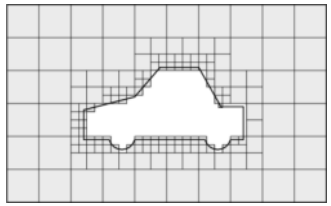


source: https://cfd.direct/openfoam/user-guide/v6-snappyhexmesh/

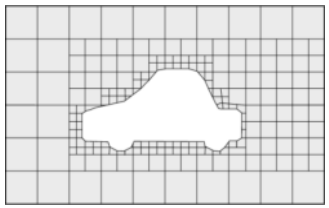
snappyHexMesh: **workflow**



background mesh

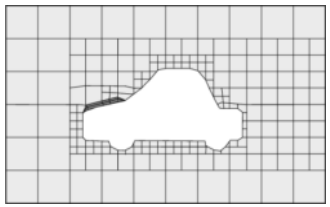


castellated mesh



snapped mesh

modify point displacements in the error region until quality criteria are matched



layered mesh

modify point displacements in the error region until quality criteria are matched

exit

Mesh generation with snappyHexMesh

snappyHexMesh generates a mesh by refining background cells,

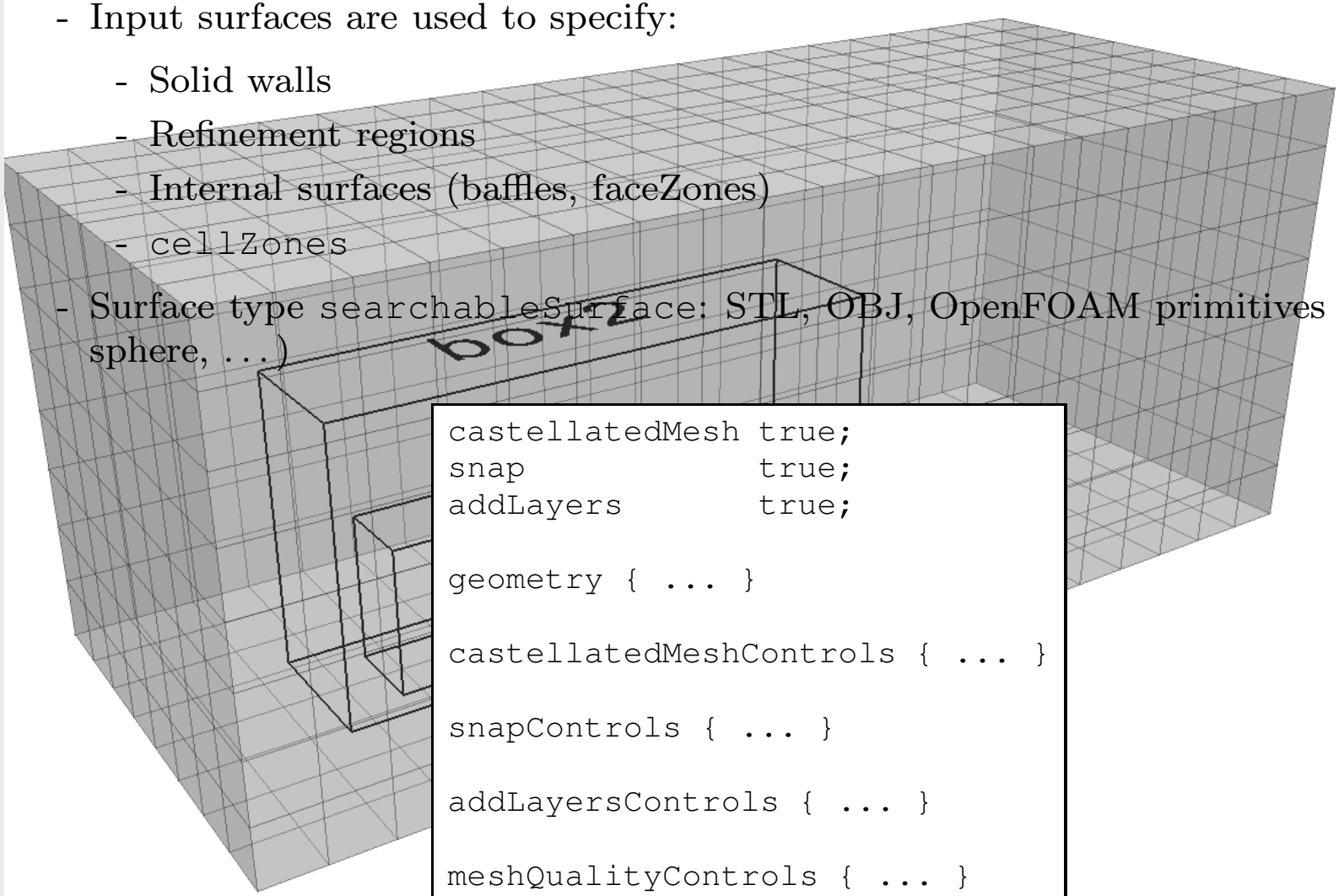
- removing unwanted cells (e.g. outside the geometry) - castellated mesh
- smoothing boundaries and snapping vertices to geometry surfaces
- and inserting layers of cells

Configuration and settings are included in the system/snappyHexMesh-Dict file, that contains:

- 3 switches to control individual meshing processes
- 3 subdictionaries, one for each meshing process, e.g. snapControls
- a geometry subdictionary for the surface geometry used in the meshing
- a meshQualityControls subdictionary for the quality criteria

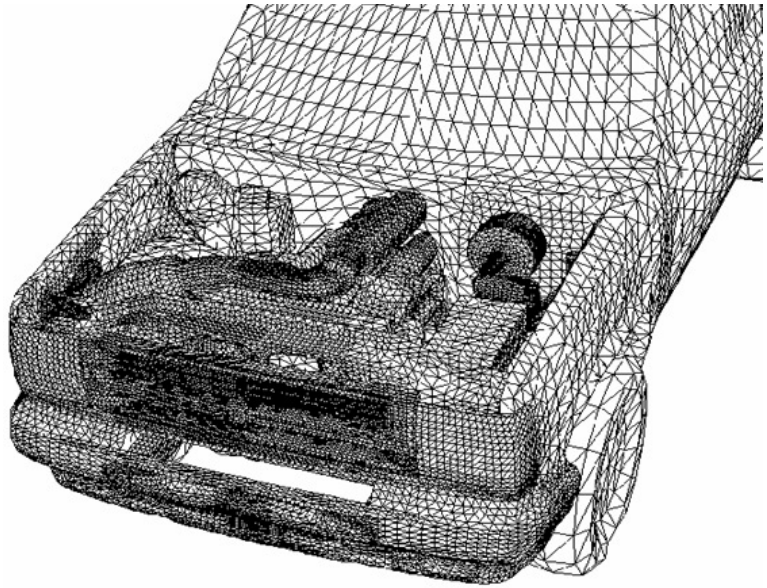
Input geometry (surface mesh)

- Input surfaces are used to specify:
  - Solid walls
  - Refinement regions
  - Internal surfaces (baffles, faceZones)
  - `cellZones`
- Surface type `searchableSurface`: STL, OBJ, OpenFOAM primitives (box, sphere, ...)




```
castellatedMesh true;
snap            true;
addLayers       true;

geometry { ... }

castellatedMeshControls { ... }

snapControls { ... }

addLayersControls { ... }

meshQualityControls { ... }
```

The STereo Lithography (STL) file format



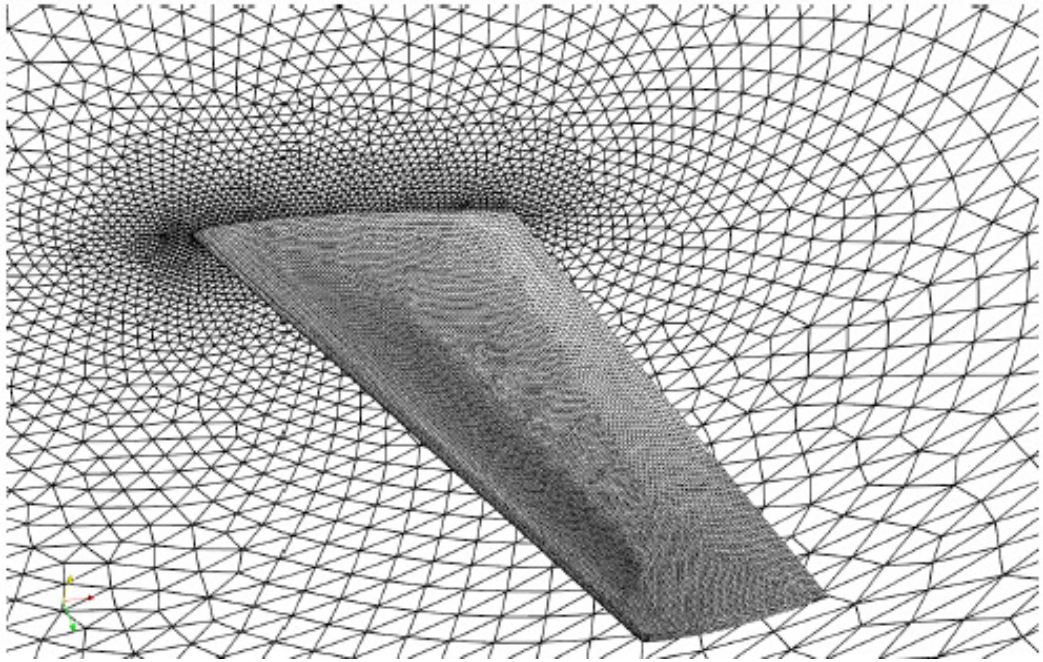
STL stands for STereo Lithography file format;

- it is a standard, open format used in many fields (e.g. 3D printers)
- can be exported by nearly any CAD/meshing application
- Surfaces are described using **triangulation**, rather than analytical functions (like IGES or STEP).
- Files can be written in `binary` or **ASCII** (i.e. plain-text) format.
- Different surfaces in the same file are known as **SOLIDS**

STL file format: structure

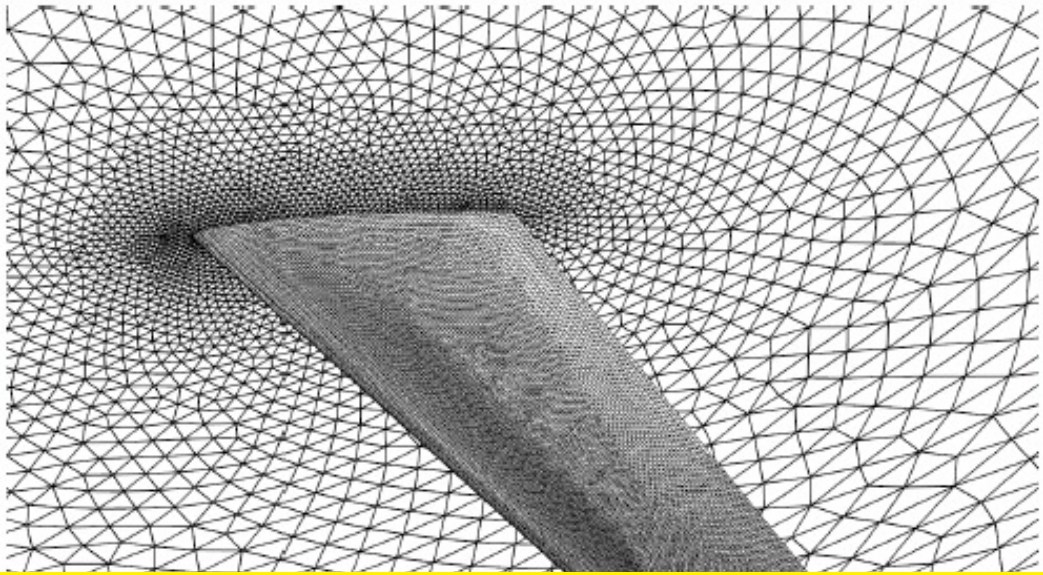

```
solid BOTTOMFACES
facet normal -0.000280212 -0.00140977 -0.999999
  outer loop
     vertex -0.00547137 0.0211336 -0.00107753
     vertex -0.00641118 0.0213204 -0.00107753
     vertex -0.00591375 0.0217819 -0.00107832
  endloop
endfacet
facet normal -9.23665e-05 -0.00128159 -0.999999
  outer loop
     vertex -0.00547137 0.0211336 -0.00107753
     vertex -0.00591375 0.0217819 -0.00107832
     vertex -0.00502158 0.0217176 -0.00107832
  endloop
endfacet
...
endsolid BOTTOMFACES
solid TOPFACES
...
endsolid TOPFACES
```

Input geometry: requested features



- The surface must be **closed**:
  - it may extend its open ends beyond the boundaries of the base mesh;
  - gaps in the surface must be possibly avoided
- the surface must have **no degenerated triangles**: use `surfaceCheck` utility
- face normals must be consistent: a sudden change in surface normal is seen as a feature edge;
- STL solids will become final mesh patches
- open surfaces can be used to specify `faceZones`, extra refinement regions

Input geometry: requested features



Using **clean**, **smooth surfaces** can let you save a lot of time later on!

- The surfa
  - it may mesh;
  - gaps i
- the surface must have **no degenerated triangles**: use `surfaceCheck` utility
- face normals must be consistent: a sudden change in surface normal is seen as a feature edge;
- STL solids will become final mesh patches
- open surfaces can be used to specify `faceZones`, extra refinement regions

The `surfaceCheck` utility

- Surface files must be in `<caseFolder>/constant/triSurface`

- Similarly to `checkMesh`, it is possible to assess the quality of a triangulated surface:

  `surfaceCheck`

- Pay attention to the bounding box: lengths are in meters in OpenFOAM!

```
Reading surface from "ahmedBody.stl" ...

Statistics:
Triangles    : 94186
Vertices     : 47440
Bounding Box : (9.43285e-7 0.2 -195.251) (1044 194.913 144.099)
```

- If necessary, you can use `surfaceTransformPoints` to scale down the geometry:

```
    surfaceTransformPoints -scale "(1e-3 1e-3 1e-3)" <inFile> <outFile>
```

The `surfaceConvert` utility

- the `surfaceConvert` utility converts surfaces from one data format to another
- available formats are:

File ext	Format
`.ftr`	OpenFOAM triangulated forma
`.stl`	Stereolithography (STL) ASCII
`.stlb`	Stereolithography (STL) binary
`.obj`	Wavefront object (OBJ)
`.off`	3D Object File Format (OFF)
`.vtk`	Visualization Toolkit (VTK)
`.tri`	Triangle format
`.ac`	Inivis AC3D
`.smesh`	TetGen surface mesh format
`.dx`	OpenDX format

Note: the `VTK` format includes a "`region`" field that can be used to color patches. To convert surface grids into VTK, type:

```
surfaceConvert <fileName>.obj <outFile>.vtk
```

This can be technically done also with STL file formats, if patches are explicitly labelled as "`solid`".

Note: the `#include` directive in dictionaries

- The `#include` syntax includes one input file from another
- The included file can be specified by local or full path, e.g.

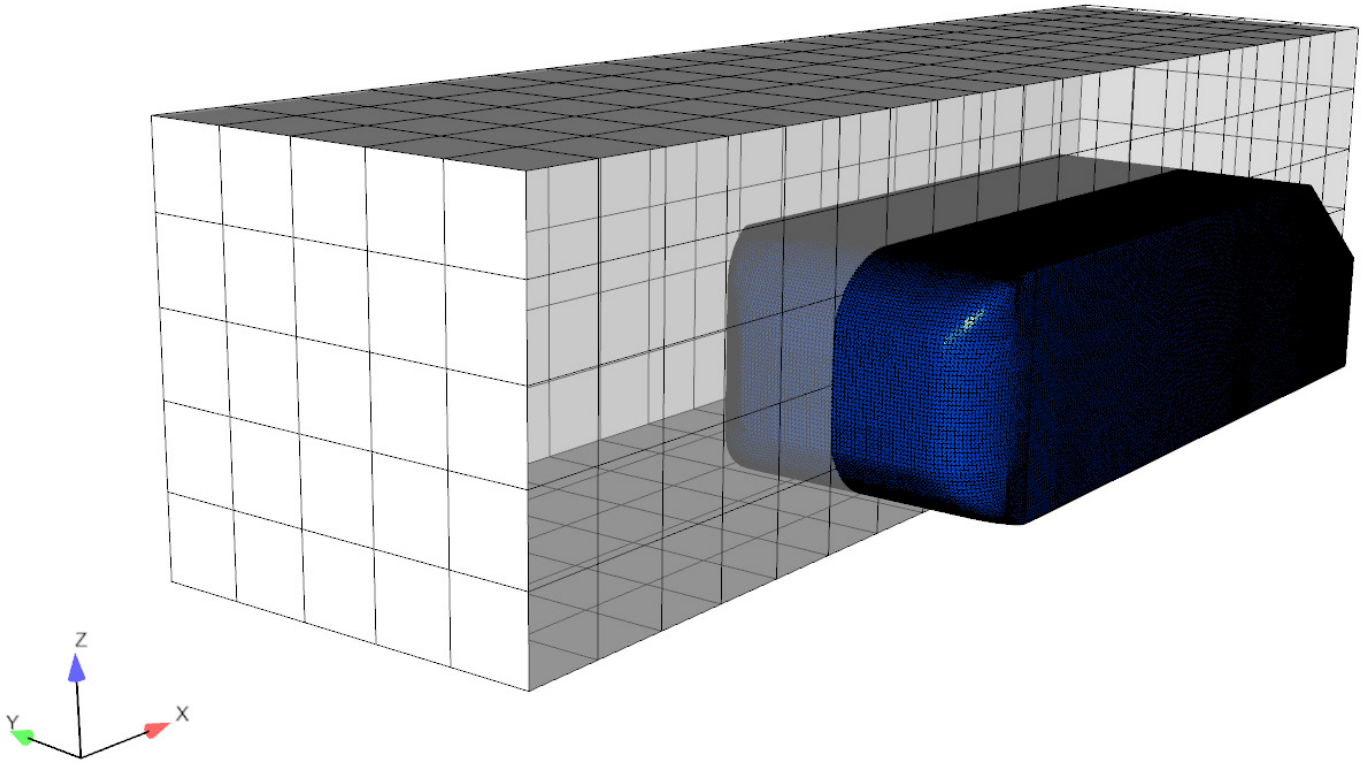
```
#include "surfaceFeatureExtractDictDefaults"
```

or

```
#include "$FOAM_CASE/system/surfaceFeatureExtractDictDefaults"
```

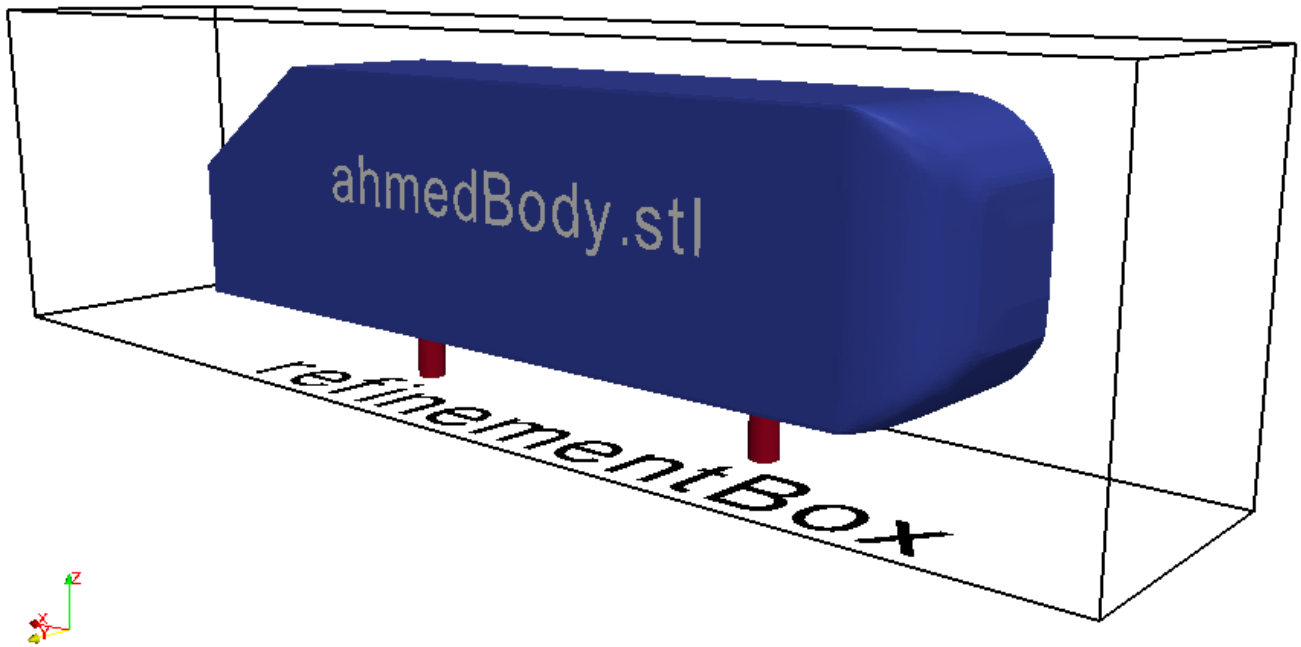
- The code interprets the `$FOAM_CASE` environment variable as the case and system environment variables, e.g. `$WM_PROJECT_DIR`

Example of Application: the Ahmed body



1. Define the geometry

- Each STL solid will be a patch in the final mesh (ASCII format required!)
- Solid name in the file can be redefined
- Define also refinement boxes, etc. . .

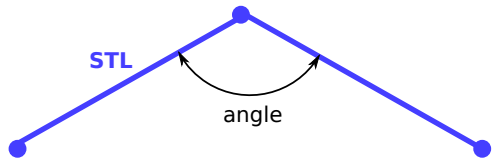


Explicit feature extraction (optional)

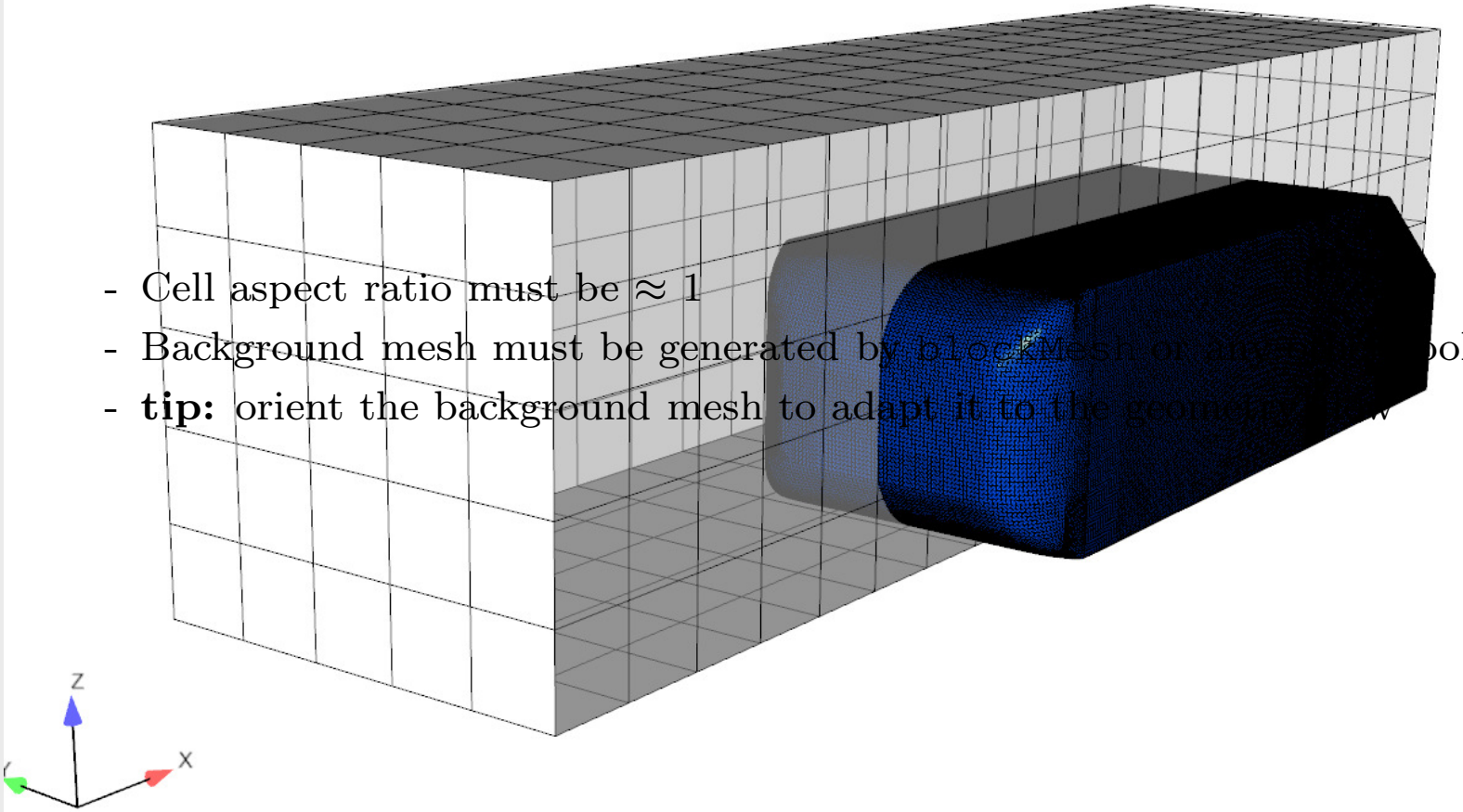
Explicit feature extraction (optional):

- you can use ParaView to select angle (filter: `Feature Edges`)
- you can use ParaView to load `edgeMesh.obj`
- use the `surfaceFeatureExtract` utility

```
ahmedBody.stl
{
    extractionMethod       extractFromSurface;

    extractFromSurfaceCoeffs
    {
        includedAngle      150;
        geometricTestOnly  yes;
    }
    // for post-processing
    writeObj               yes;
}
```



2. Generate the background mesh



- Cell aspect ratio must be ≈ 1
- Background mesh must be generated by blockMesh or another tool
- **tip:** orient the background mesh to adapt it to the geometry

Generate the background mesh

The utility `surfaceCheck` can be used to generate automatically the first lines of a `blockMeshDict`:

```
surfaceCheck -blockMesh <fileName>
```

being `fileName=constant/triSurface/ahmedBody.stl`. The command returns an output on the screen, that you can **copy/paste in the** `blockMeshDict` **file**.

```
// ...
// blockMeshDict info

vertices
(
    (9.43285e-10 0.0002 -0.195251)
    (1.044 0.0002 -0.195251)
    //...
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (10 10 10) simpleGrading (1 1 1)
);

edges
();
patches
();
// end blockMeshDict info
// ...
```

Generate the background mesh

Note: external patches (generated by `blockMesh` should have descriptive names and correct types!

```
boundary
(
    frontAndBack
    {
        type patch;
        faces
        (
            (blk0 2)
            (blk0 3)
        );
    }

    inlet  { ... }
    outlet { ... }

    lowerWall
    {
        type wall;
        faces
        (
            (blk0 4)
        );
    }

    upperWall { ... }
);
```

3. Castellated Mesh Generation

Max Number of Cells Per Processor (maxLocalCells): specifies the maximum number of cells per processor during the refinement phase. If local number of cells is greater than Max Number of Cells Per Processor (`maxLocalCells`) on any processor the algorithm switches from refinement followed by balancing (current method) to balancing before refinement (weighted).

Global Max Number of Cells (`maxGlobalCells`): specifies the maximum overall number of cells during the refinement phase. Refinement process will terminated immediately when the overall cells during refinement exceed this number. In this case it may be that the refinement level might not complete. (for example some region refinements are not carried out) → do not exceed 15M cells every 8 cores, unless you have lot of RAM memory!

Cell Limit for Refinement Termination (`minRefinementCells`): specifies the minimum number of cells during the refinement phase. For instance the surface refinement algorithm might spend lots of iterations refining just a some cells. This setting will cause refinement to stop if cells to be refined are less than specified value Note: at least one iteration will be done, unless the number of cells to refine is 0.

Castellated Mesh Generation

Max Processor Load Imbalance (`maxLoadUnbalance`): specifies the maximum unbalance for processors in parallel meshing process. A value of 0 forces to balance always.

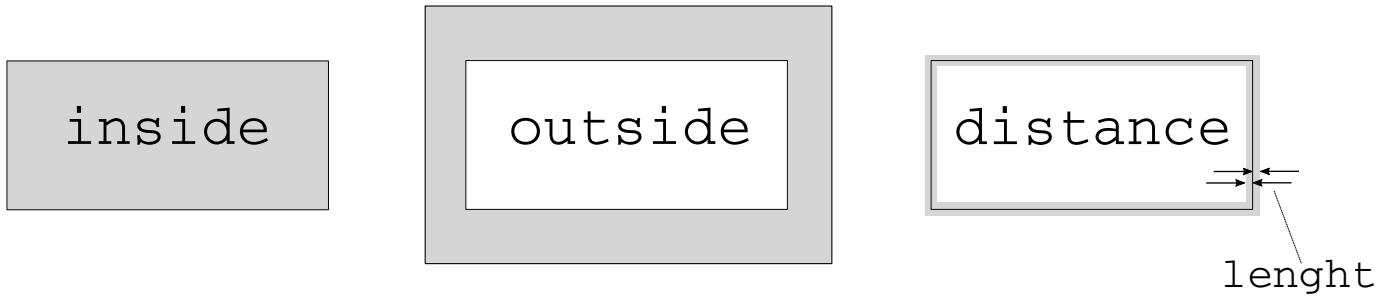
Buffer Cell Count between Refinement Levels (`nCellsBetweenLevels`): specifies the number of cell layers between different levels of refinement; 1 means that if required only a single intermediate layer is added to match the refinement difference. Larger values like 5, will add more intermediate cell layers to gradually cover the refinement difference but greatly effect the mesh size.

Castellated Mesh Generation


```
refinementRegions
{
  refinementBox
  {
    mode distance;      // options: inside/outside/distance
    length 0.001;
    levels ((1e15 3)); // (entry 1e15 ignored)
  }
}
```

`refinementRegions` describes refinement within a volume region:

- user defines the name of the surface geometry, refinement mode and `levels`
- `mode` specifies where the refinement should occur:



Castellated Mesh Generation

`refinementSurfaces`: cells are refined if intersected by the STL surface.

- the user specifies the name of the surface geometry and min/max refinement level
- background level = 0;

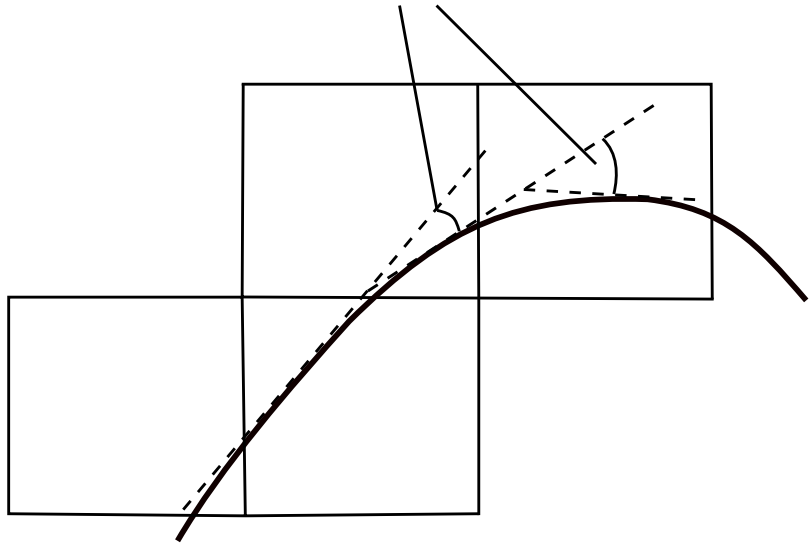
```
refinementSurfaces
{
    ahmedBody          // name of surface geometry
    {
        level (0 0);  // (min/max) refinement level on ALL surfaces

        regions
        {
            body { level (3 4); }   // refinement level specific for the region
            ...
```

Castellated Mesh Generation

`refinementSurfaces`: cells are refined if intersected by the STL surface.

- the user specifies the name of the surface geometry and min/max refinement level
- background level = 0;
- min level applied at least to all cells on surface
- refines to max, if surface orientation in adjacent cells > `resolvedFeatureAngle`

if $\theta >$ `resolveFeatureAngle` refine further up to `max` level

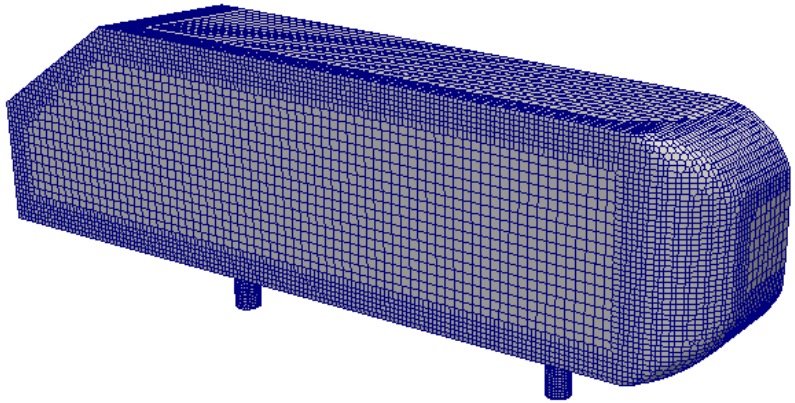


source: http://www.training.prace-ri.eu/uploads/tx_pracetmo/snappyHexMesh.pdf

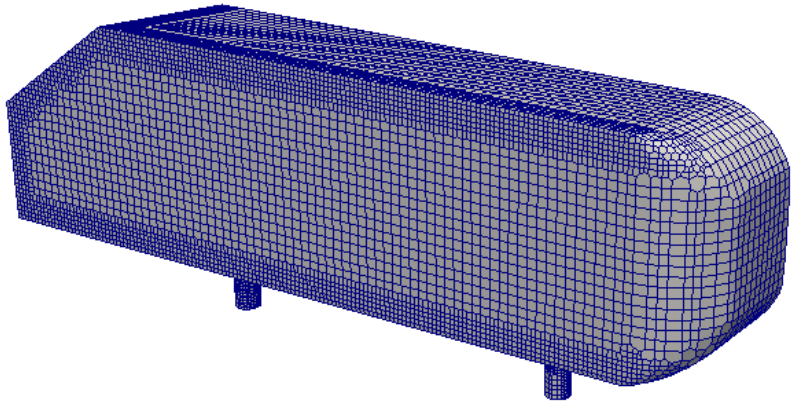
Castellated Mesh Generation

Select `featureAngle`:

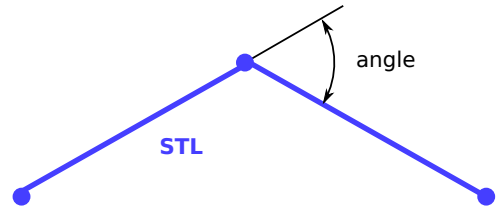
If angle > `featureAngle`, cells are refined up to the maximum level.



angle = 10°



angle = 50°

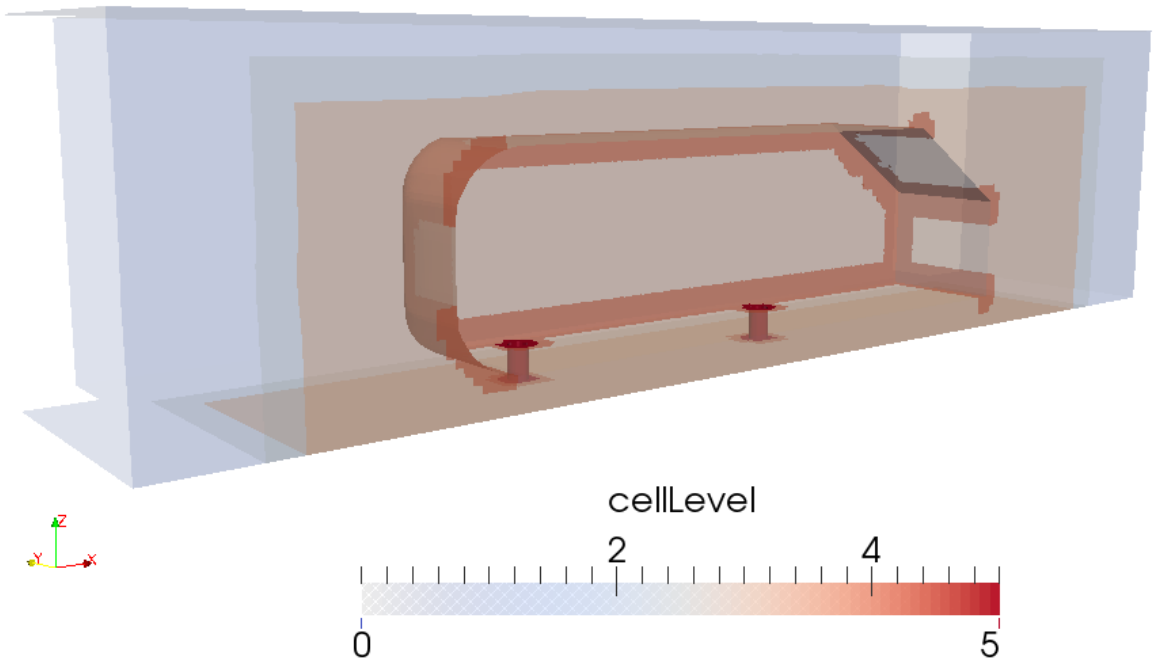


Castellated Mesh Generation

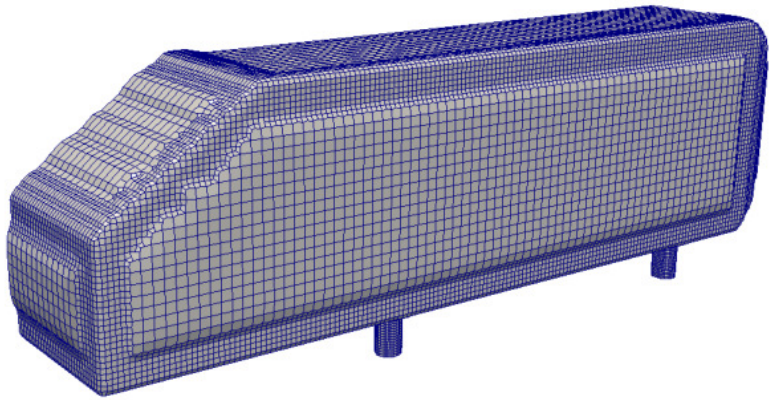
Check `refinementLevels`:

You can save the scalarField **cellLevels** for later post-processing:

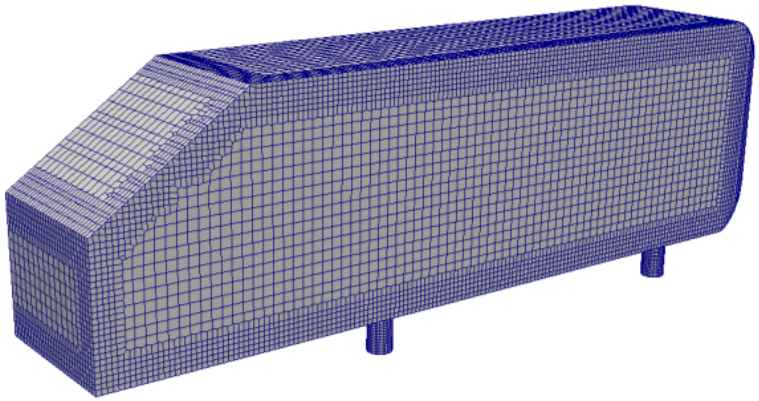
```
writeFlags
(
    scalarLevels
);
```



4. Snap the mesh



tolerance = 0.1



tolerance = 2.0

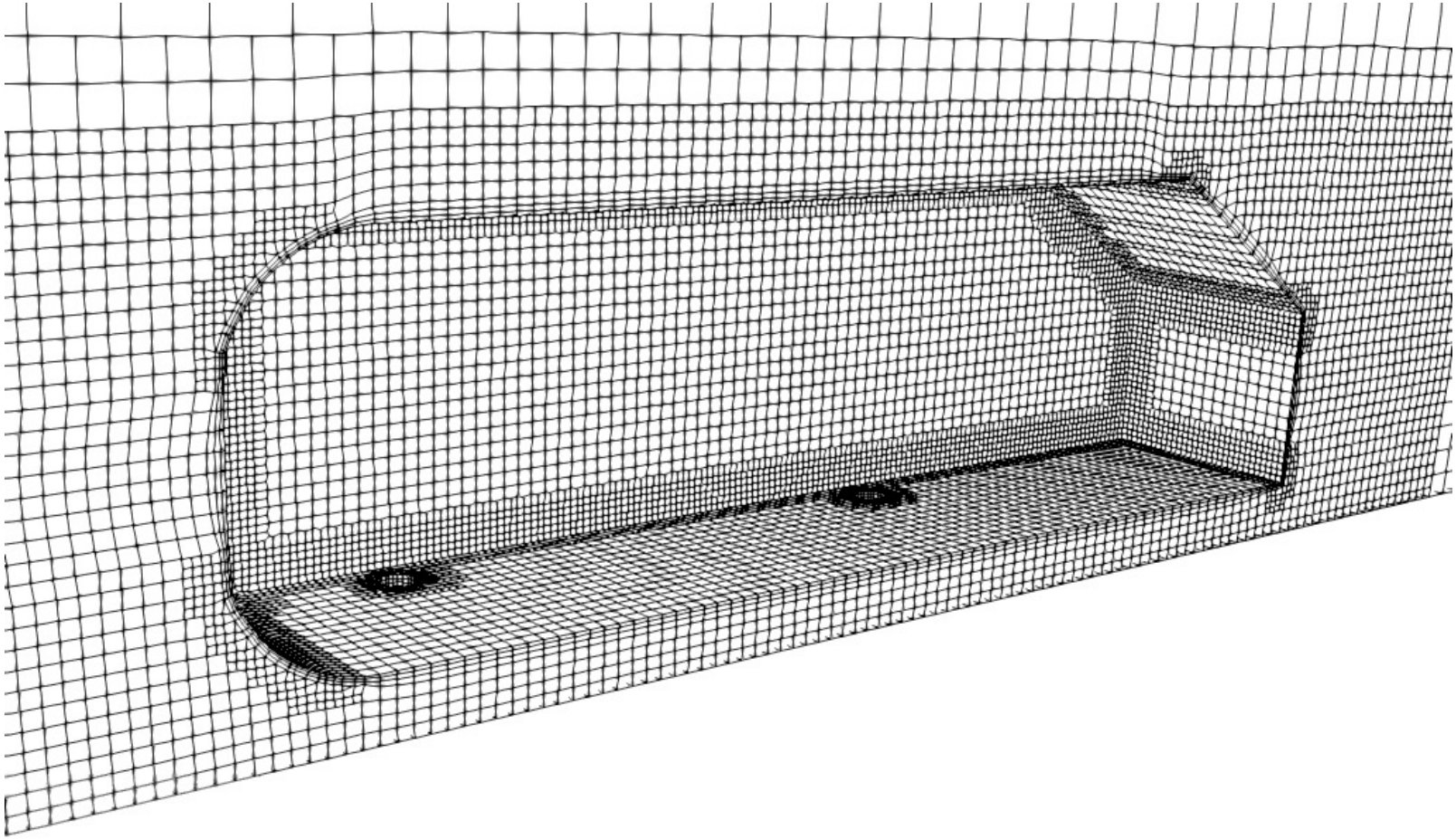
```
// Settings for the snapping.
snapControls
{
  nSmoothPatch      3;
  tolerance         2.0;
  nSolveIter       30;
  nRelaxIter        5;
  nFeatureSnapIter 10;

  implicitFeatureSnap false;
  explicitFeatureSnap true;
  multiRegionFeatureSnap false;
}
```

Snap controls: tips

- Default values are OK for most situations
- Increasing the number of iterations can increase a lot the generation time but may produce a mesh with less errors
- Explicit feature recognition produce always best results but requires additional steps

```
// Settings for the snapping.
snapControls
{
  nSmoothPatch      3;
  tolerance         2.0;
  nSolveIter       30;
  nRelaxIter        5;
  nFeatureSnapIter 10;

  implicitFeatureSnap false;
  explicitFeatureSnap true;
  multiRegionFeatureSnap false;
}
```

5. Adding wall layers



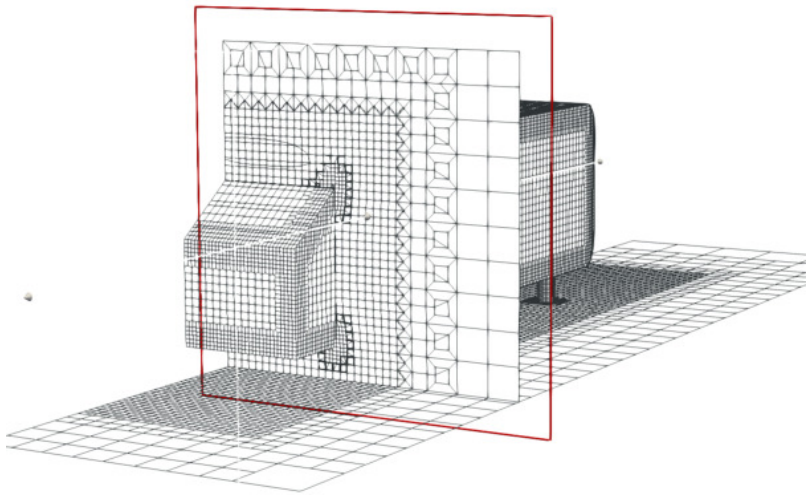
5. Adding wall layers

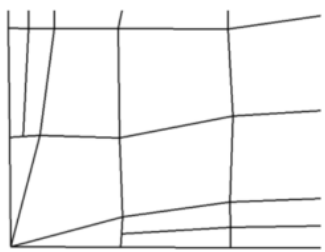

```
relativeSizes true;

// Per final patch
layers
{
    body
    {
        nSurfaceLayers 3;
    }
    legs
    {
        nSurfaceLayers 3;
    }
}

expansionRatio 1.3;
finalLayerThickness 0.4;

minThickness 0.1;
```

- Thickness in absolute units or relative to cell size
- Different methods to specify thickness:
  - expansionRatio and finalLayerThickness (cell nearest internal mesh)
  - expansionRatio and firstLayerThickness (cell on surface)
  - overall thickness and firstLayerThickness
  - overall thickness and finalLayerThickness
  - overall thickness and expansionRatio

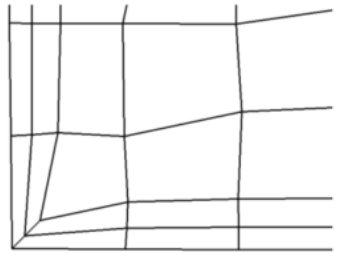
6. Wall layers: advanced settings


```
nGrow 0;

featureAngle 60;
slipFeatureAngle 30;

nRelaxIter 3;
nSmoothSurfaceNormals 1;
nSmoothNormals 3;
nSmoothThickness 10;

maxFaceThicknessRatio 0.5;
maxThicknessToMedialRatio 0.3;
minMedianAxisAngle 90;

nBufferCellsNoExtrude 0;
nLayerIter 50;
nRelaxedIter 20;
```





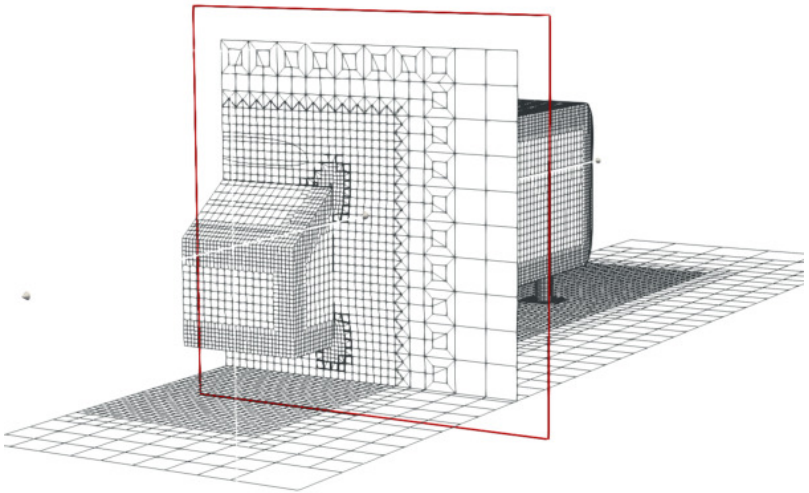
featureAngle 45

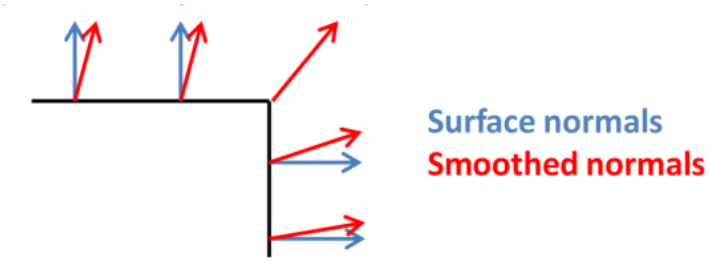


featureAngle 180

6. Wall layers: advanced settings

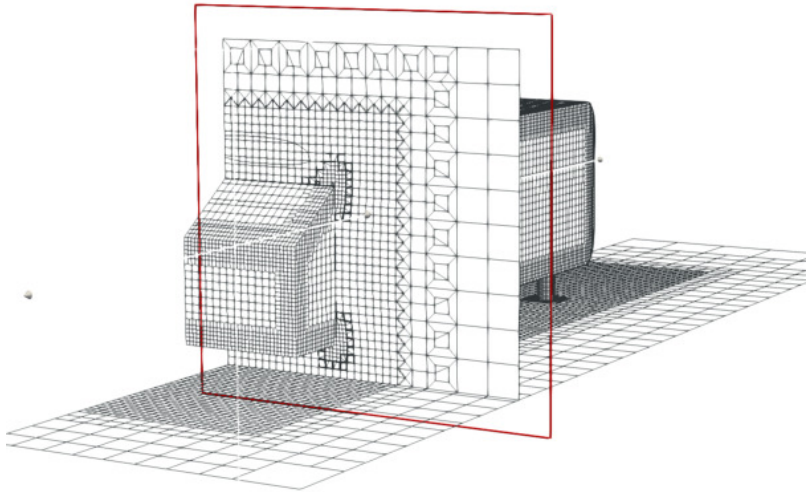

```
nGrow 0;

featureAngle 60;
slipFeatureAngle 30;

nRelaxIter 3;
nSmoothSurfaceNormals 1;
nSmoothNormals 3;
nSmoothThickness 10;

maxFaceThicknessRatio 0.5;
maxThicknessToMedialRatio 0.3;
minMedianAxisAngle 90;

nBufferCellsNoExtrude 0;
nLayerIter 50;
nRelaxedIter 20;
```

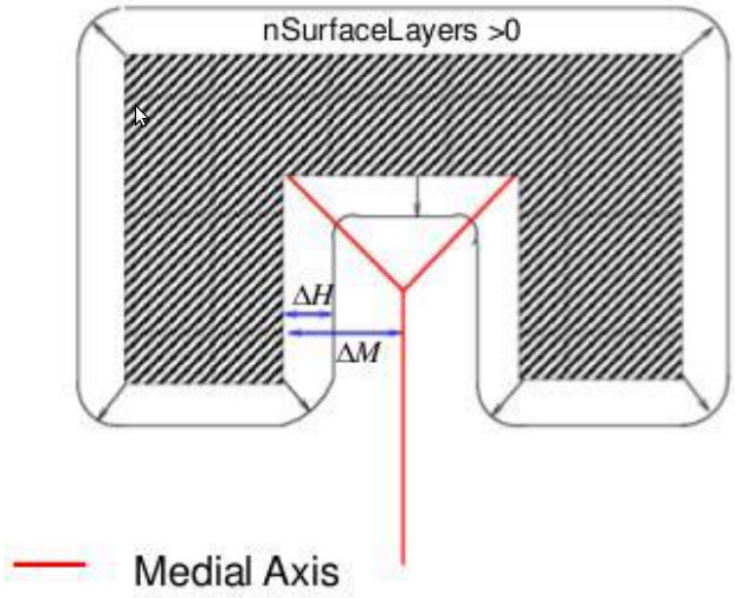




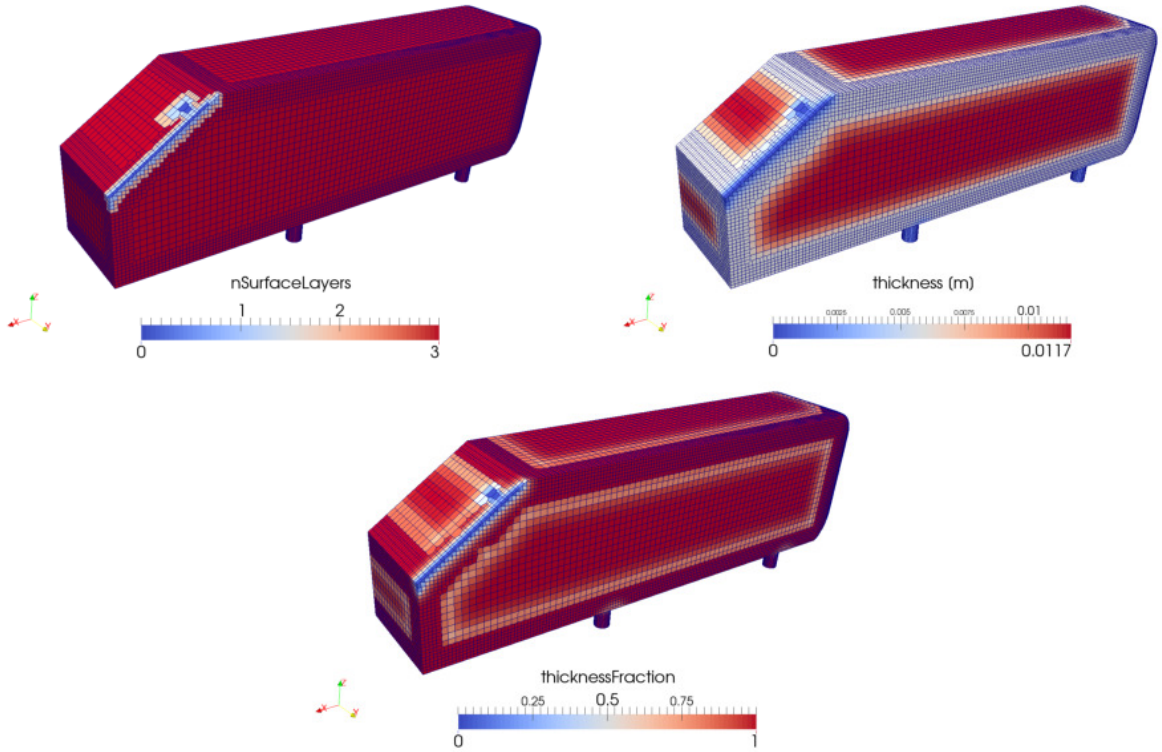
6. Wall layers: advanced settings


```
nGrow 0;

featureAngle 60;
slipFeatureAngle 30;

nRelaxIter 3;
nSmoothSurfaceNormals 1;
nSmoothNormals 3;
nSmoothThickness 10;

maxFaceThicknessRatio 0.5;
maxThicknessToMedialRatio 0.3;
minMedianAxisAngle 90;

nBufferCellsNoExtrude 0;
nLayerIter 50;
nRelaxedIter 20;
```





7. Wall layers: tips



`layerFields` can help you to find out what is wrong

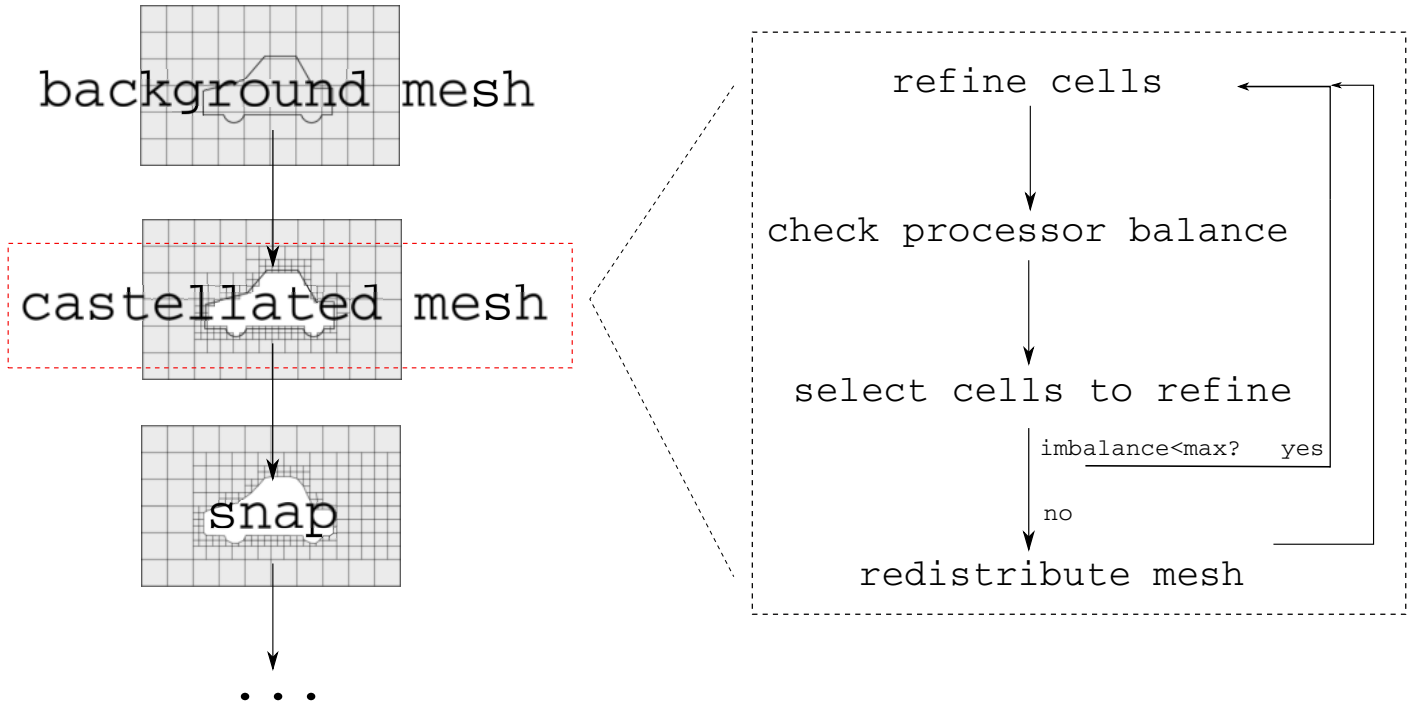
```
writeFlags
(
    scalarLevels
    layerFields
);
```

8. Mesh quality controls


```
meshQualityControls
{
    maxNonOrtho 65;
    maxBoundarySkewness 20;
    maxInternalSkewness 4;
    maxConcave 80;
    minVol 1e-13;
    minTetQuality 1e-15;
    minArea -1;
    minTwist 0.02;
    minDeterminant 0.001;
    minFaceWeight 0.05;
    minVolRatio 0.01;
    minTriangleTwist -1;
    //minVolCollapseRatio 0.1;

    nSmoothScale 4;

    errorReduction 0.75;

    relaxed
    {
        maxNonOrtho 75;
        maxBoundarySkewness 25;
        maxInternalSkewness 8;
    }
}
```

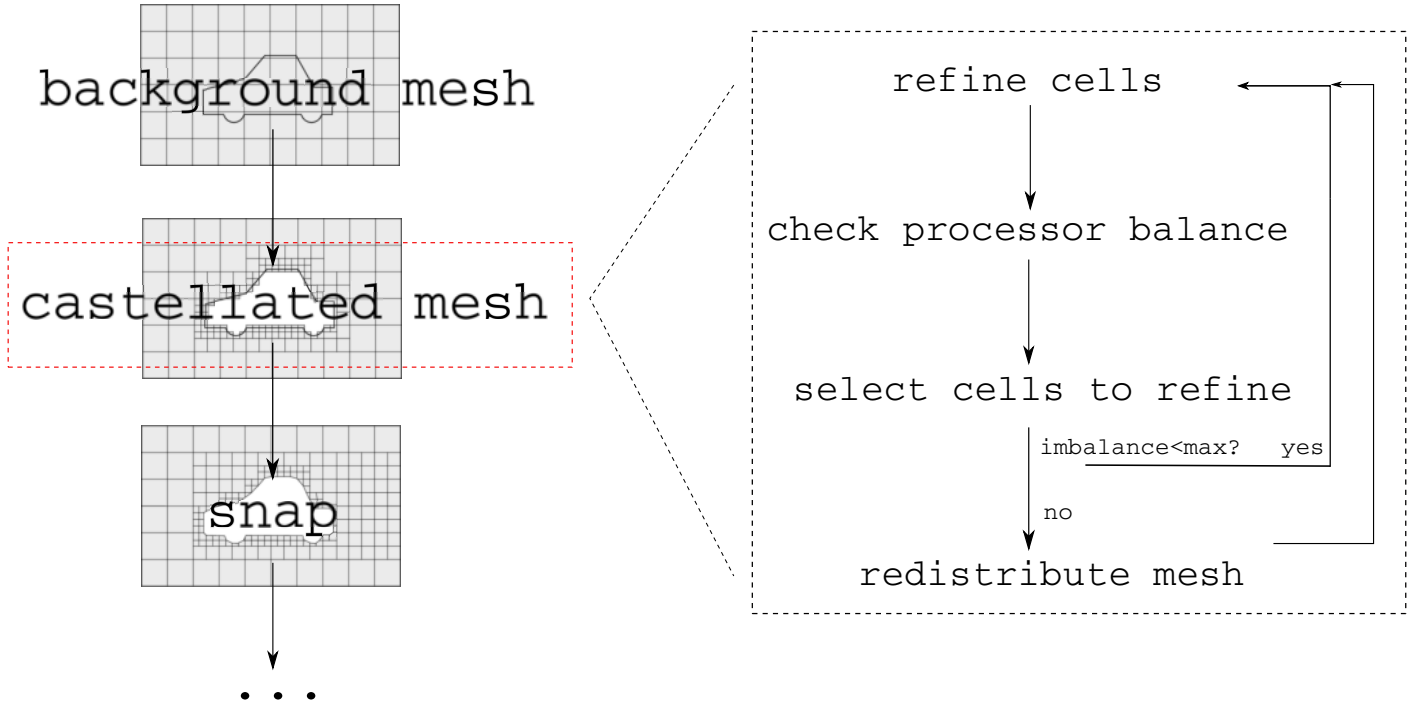
- If a constraint is not fulfilled, the action (snap, layer addition) is undone and another trial is made with a relaxed displacement
- Too loose constraints may result in a poor mesh
- Too tight constraint will result in a long process and, possibly, in no result (esp. layers)
- If the layer loop exits because `nLayerIter` has been reached, additional `nRelaxedIter` with looser quality constraints are performed

snappyHexMesh on parallel architectures



- snappyHexMesh can run on any number of processors, thus achieving **parallel mesh generation**
- Exploits **ptscotch** parallel decomposition library
- Mesh needs to be redistributed so long as cells are refined, to maintain a balanced decomposition

snappyHexMesh on parallel architectures



- Mesh redistribution is very time-consuming: a tradeoff must be sought
- Inter-processor communication is required also during snap and layer addition

Conclusions

`snappyHexMesh` is a tool for **automatic** mesh generation.

- it is **not** a CAD tool: supplied geometry must be already 'clean'
- algorithm parameters must be supplied by the user (for some of them, default values are OK)
- Perfect for parametric/optimization studies

`snappyHexMesh` works in parallel

- very convenient for large cases (up to 100M cells)
- works on any cluster with MPI architecture
- mesh redistribution is an actual bottleneck
- drawback: non-negligible memory consumption

The resulting generated mesh is hexa-dominant, not-oriented; it favors a very good performance in combination with the numerical solvers of OpenFOAM.

snappyHexMesh: pro and cons

- **What are the main advantages of** snappyHexMesh**?**
  - It is an open-source tool capable of dealing with complex geometries;
  - it generates hexa-dominant mesh;
  - it works in parallel and favors fast generation of large grids;
  - it is very easy to configure in automatic workflows, to carry out parametric studies/optimization workchains;
  - it is a great tool to for fast generation of complex geometries.
- **Are there any drawbacks?**
  - a graphical interface is missing (though some alternatives exist)
  - it is not the best option to simulate wall-bounded flows where oriented grids are needed.

Thank you for your attention!

contact: federico.piscaglia@polimi.it