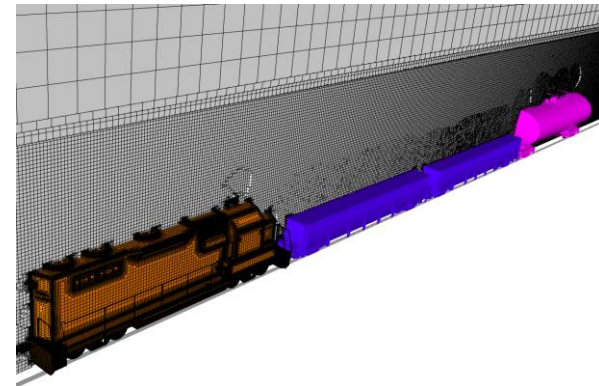
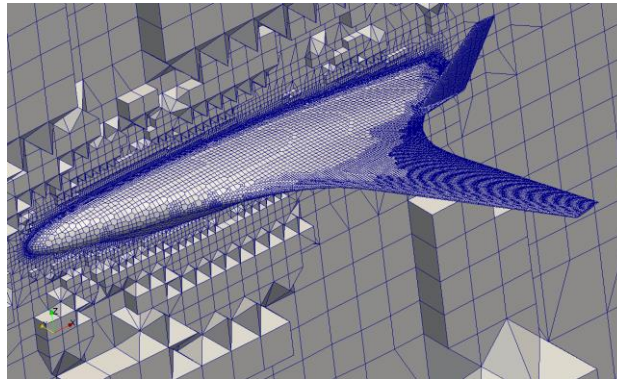
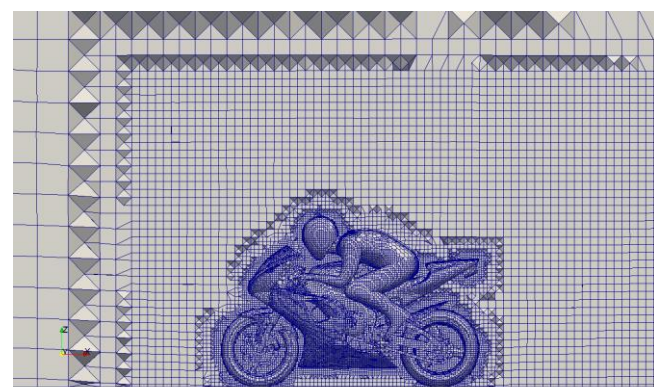


snappyHexMesh

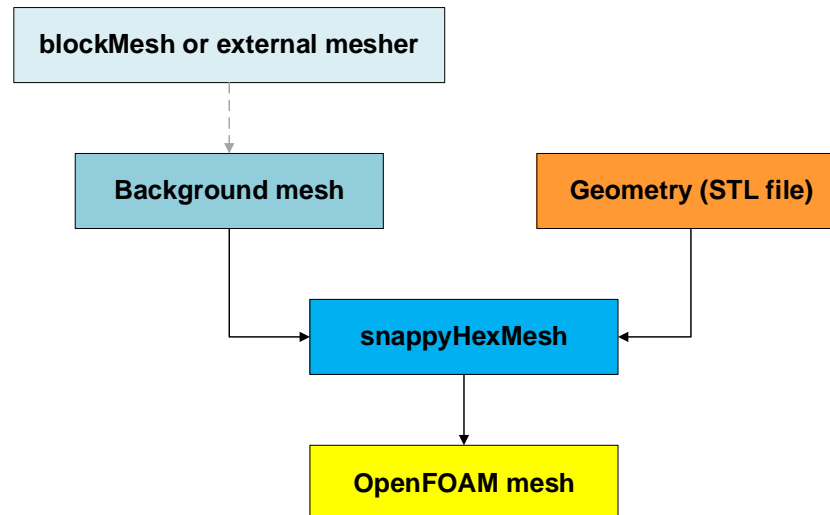
- “Automatic split hex mesher. Refines and snaps to surface.”
- For complex geometries, the mesh generation utility `snappyHexMesh` can be used.
- The `snappyHexMesh` utility generates 3D meshes containing hexahedra and split-hexahedra from a triangulated surface geometry in Stereolithography (STL) format.
- The mesh is generated from a dictionary file named `snappyHexMeshDict` located in the system directory and a triangulated surface geometry file located in the directory `constant/triSurface`.



Mesh generation using snappyHexMesh

snappyHexMesh workflow

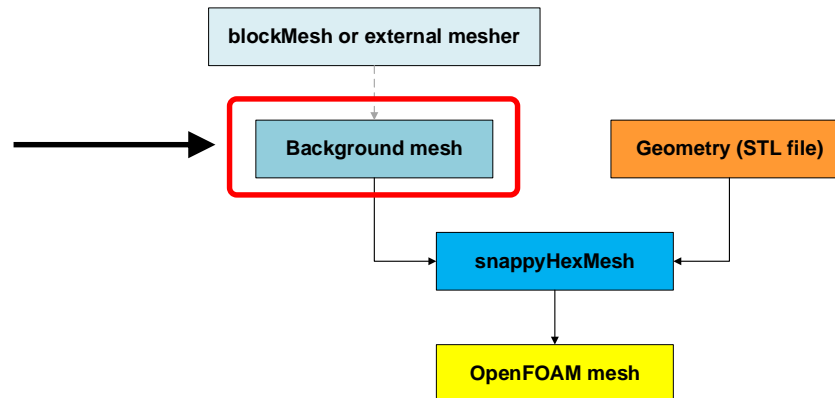
- To generate a mesh with snappyHexMesh we proceed as follows:
 - Generation of a background or base mesh.
 - Geometry definition.
 - Generation of a castellated mesh or cartesian mesh.
 - Generation of a snapped mesh or body fitted mesh.
 - Addition of layers close to the surfaces or boundary layer meshing.
 - Check/enforce mesh quality.



Mesh generation using snappyHexMesh

snappyHexMesh workflow – Background mesh

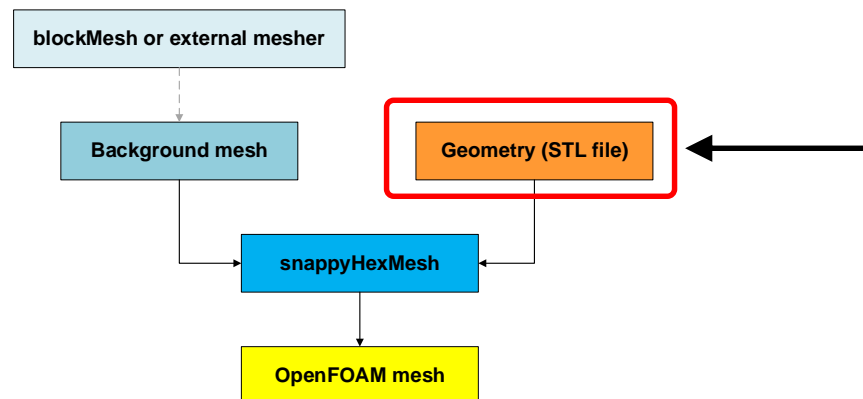
- The background or base mesh can be generated using blockMesh or an external mesher.
- The following criteria must be observed when creating the background mesh:
 - The mesh must consist purely of hexes.
 - The cell aspect ratio should be approximately 1, at least near the STL surface.
 - There must be at least one intersection of a cell edge with the STL surface.



Mesh generation using snappyHexMesh

snappyHexMesh workflow – Geometry (STL file)

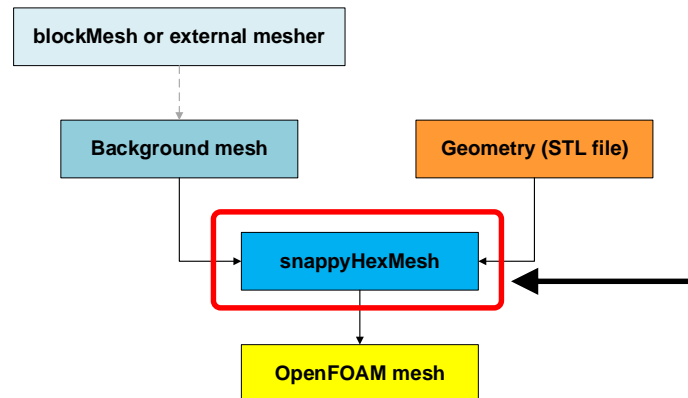
- The STL geometry can be obtained from any geometry modeling tool.
- The STL file can be made up of a single surface describing the geometry, or multiple surfaces that describe the geometry.
- In the case of a STL file with multiple surfaces, we can use local refinement in each individual surface. This gives us more control when generating the mesh.
- The STL geometry is always located in the directory **constant/triSurface**



Mesh generation using snappyHexMesh

snappyHexMesh workflow

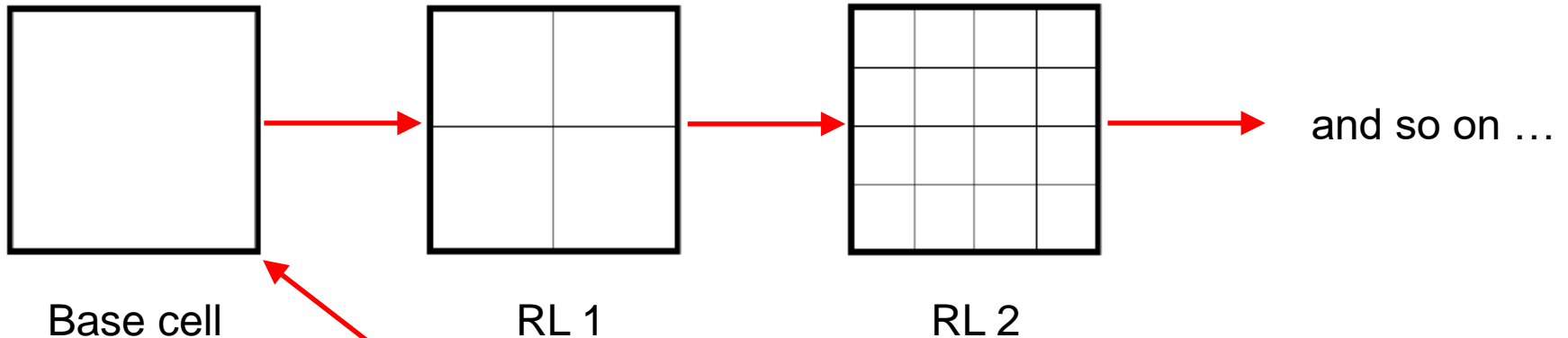
- The meshing utility `snappyHexMesh` reads the dictionary `snappyHexMeshDict` located in the directory system.
- The castellation, snapping, and boundary layer meshing steps are controlled by the dictionary `snappyHexMeshDict`.
- The final mesh should be always located in the directory **constant/polyMesh**



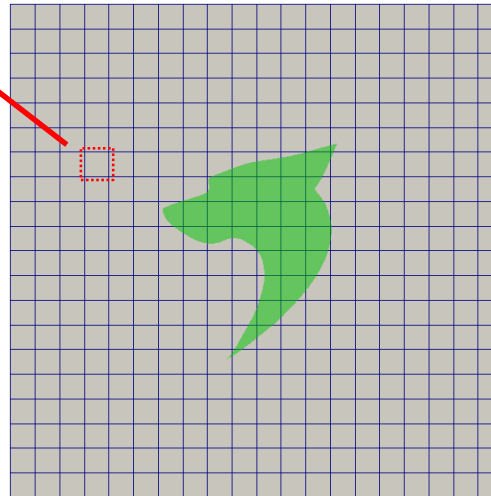
Mesh generation using snappyHexMesh

snappyHexMesh workflow

- All the volume and surface refinement is done in reference to the background or base mesh.

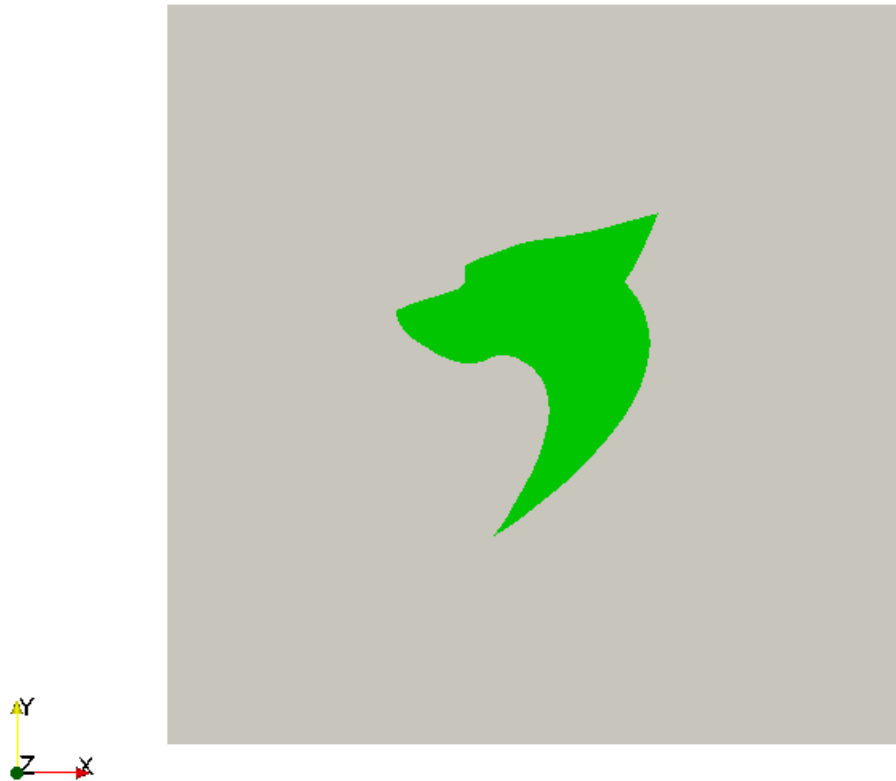


* RL = refinement level



Mesh generation using snappyHexMesh

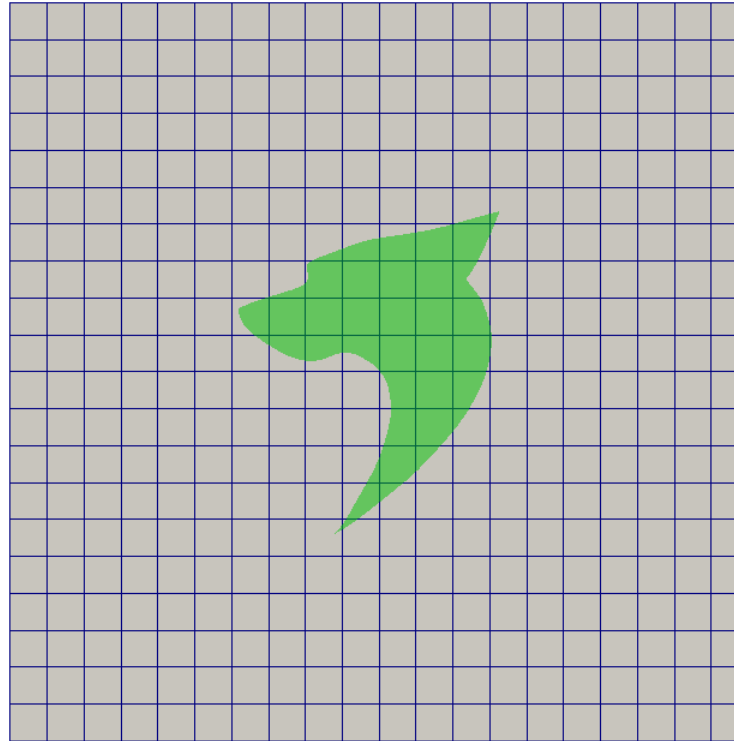
snappyHexMesh workflow



- The process of generating a mesh using `snappyHexMesh` will be described using this figure.
- The objective is to mesh a rectangular shaped region (shaded grey in the figure) surrounding an object described by a STL surface (shaded green in the figure).
- This is an external mesh (e.g. for external aerodynamics). You can also generate an internal mesh (e.g. flow in a pipe).

Mesh generation using snappyHexMesh

snappyHexMesh workflow

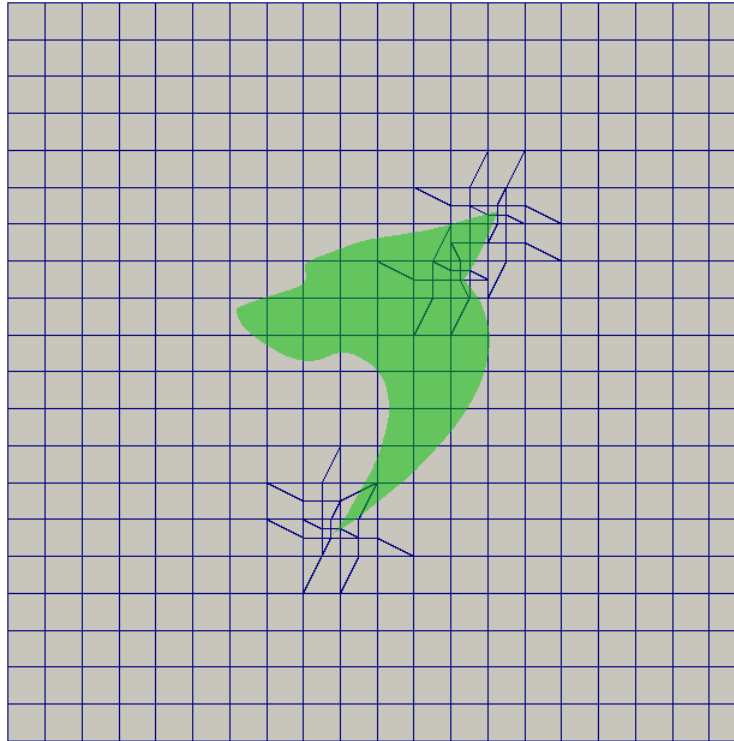


Step 1. Creating the background hexahedral mesh

- Before `snappyHexMesh` is executed the user must create a background mesh of hexahedral cells that fills the entire region as shown in the figure. This can be done by using `blockMesh` or any other mesher.
- The following criteria must be observed when creating the background mesh:
 - The mesh must consist purely of hexes.
 - The cell aspect ratio should be approximately 1, at least near the STL surface.
 - There must be at least one intersection of a cell edge with the STL surface.

Mesh generation using snappyHexMesh

snappyHexMesh workflow

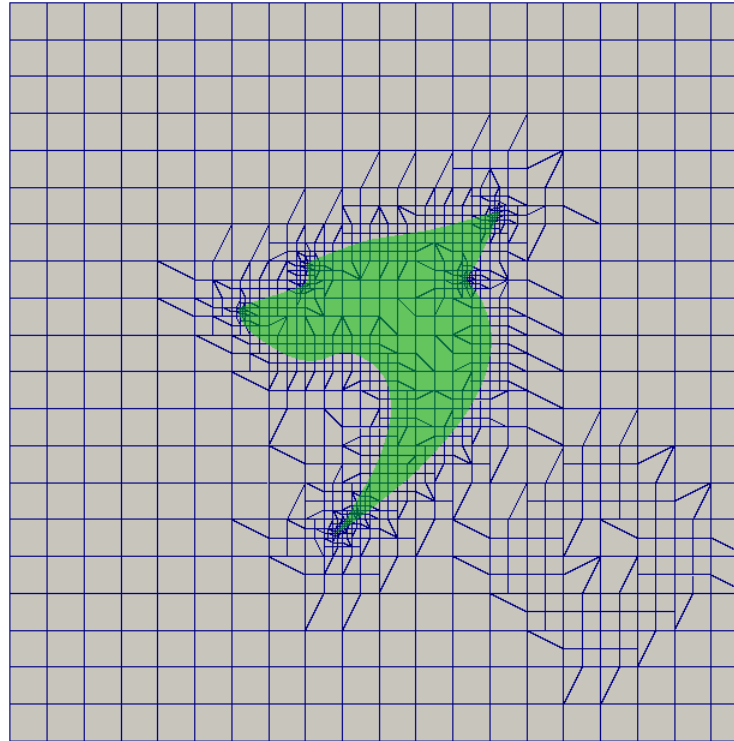


Step 2. Cell splitting at feature edges

- Cell splitting is performed according to the specification supplied by the user in the **castellatedMeshControls** sub-dictionary in the *snappyHexMeshDict* dictionary.
- The splitting process begins with cells being selected according to specified edge features as illustrated in the figure.
- The feature edges can be extracted from the STL geometry file using the utility `surfaceFeatureExtract`.

Mesh generation using snappyHexMesh

snappyHexMesh workflow

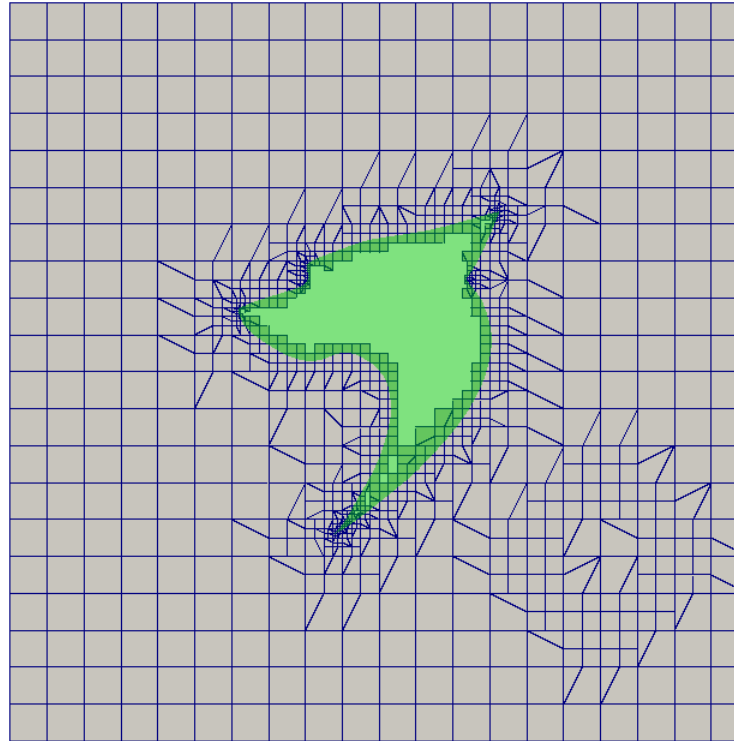


Step 3. Cell splitting at surfaces

- Following feature edges refinement, cells are selected for splitting in the locality of specified surfaces as illustrated in the figure.
- The surface refinement (splitting) is performed according to the specification supplied by the user in the **refinementMeshControls** in the **castellatedMeshControls** sub-dictionary in the *snappyHexMeshDict* dictionary.

Mesh generation using snappyHexMesh

snappyHexMesh workflow

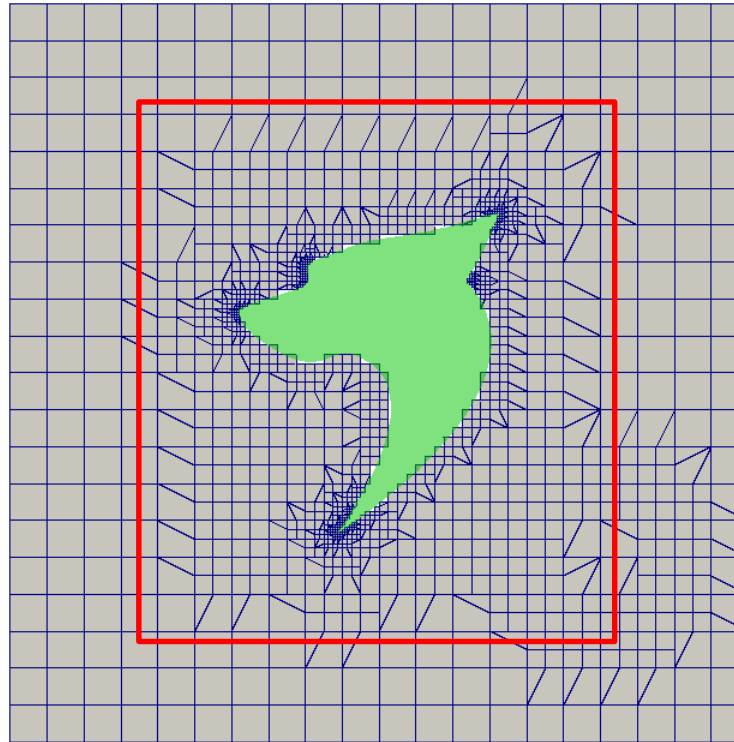


Step 4. Cell removal

- Once the feature edges and surface splitting is complete, a process of cell removal begins.
- The region in which cells are retained are simply identified by a location point within the region, specified by the **locationInMesh** keyword in the **castellatedMeshControls** sub-dictionary in the *snappyHexMeshDict* dictionary.
- Cells are retained if, approximately speaking, 50% or more of their volume lies within the region.

Mesh generation using snappyHexMesh

snappyHexMesh workflow

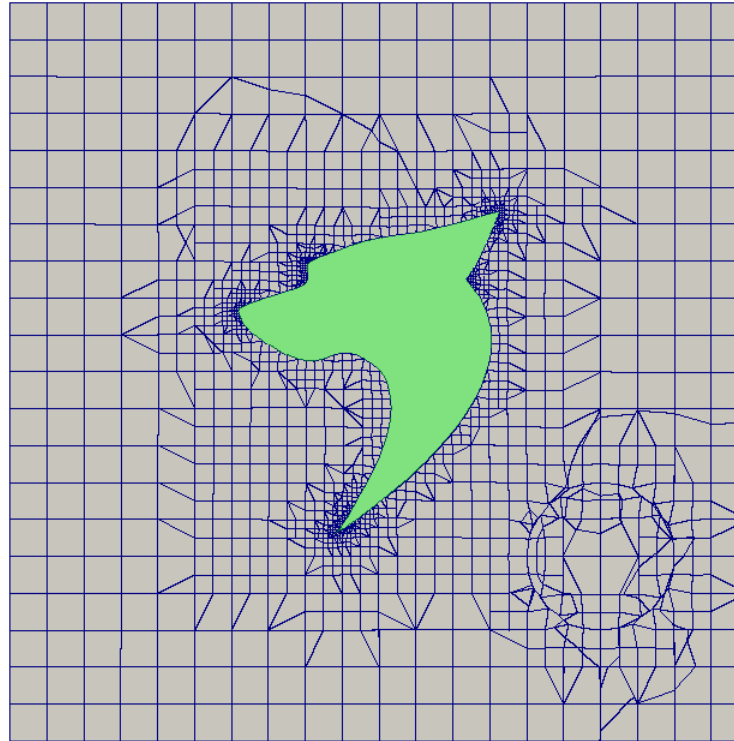


Step 5. Cell splitting in specified regions

- Those cells that lie within one or more specified volume regions can be further split by a region (in the figure, the rectangular region within the red rectangle).
- The information related to the refinement of the volume regions is supplied by the user in the **refinementRegions** block in the **castellatedMeshControls** sub-dictionary in the *snappyHexMeshDict* dictionary.
- This is a valid castellated or cartesian mesh that can be used for a simulation.

Mesh generation using snappyHexMesh

snappyHexMesh workflow

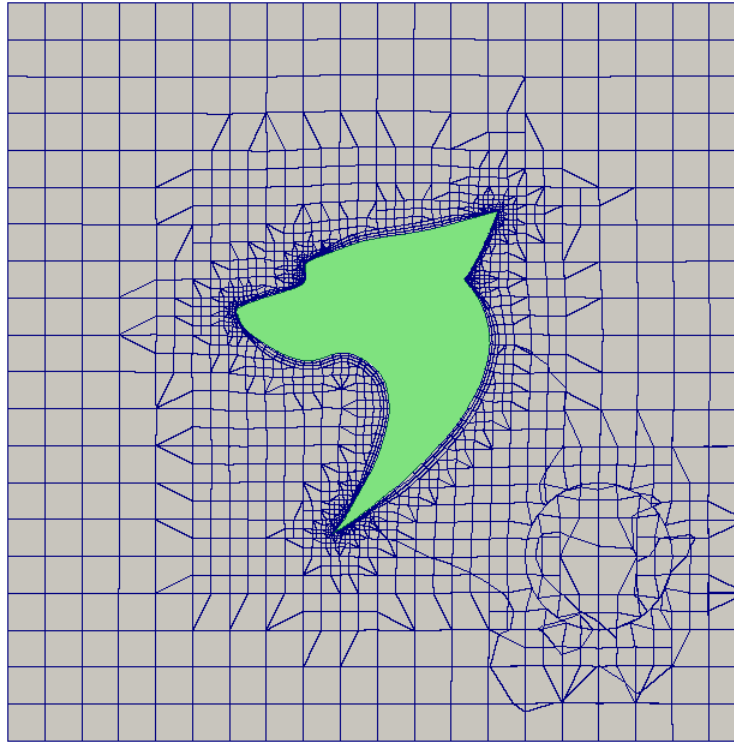


Step 6. Snapping to surfaces

- After deleting the cells in the region specified and refining the volume mesh, the points are snapped on the surface to create a conforming mesh.
- The snapping is controlled by the user supplied information in the **snapControls** sub-dictionary in *snappyHexMeshDict*.
- Sometimes, the default **snapControls** options are not enough and you will need to adjust the values to get a good mesh, so it is advisable to save the intermediate steps with a high writing precision (*controlDict*).
- This is a valid snapped or body fitted mesh that can be used for a simulation.

Mesh generation using snappyHexMesh

snappyHexMesh workflow



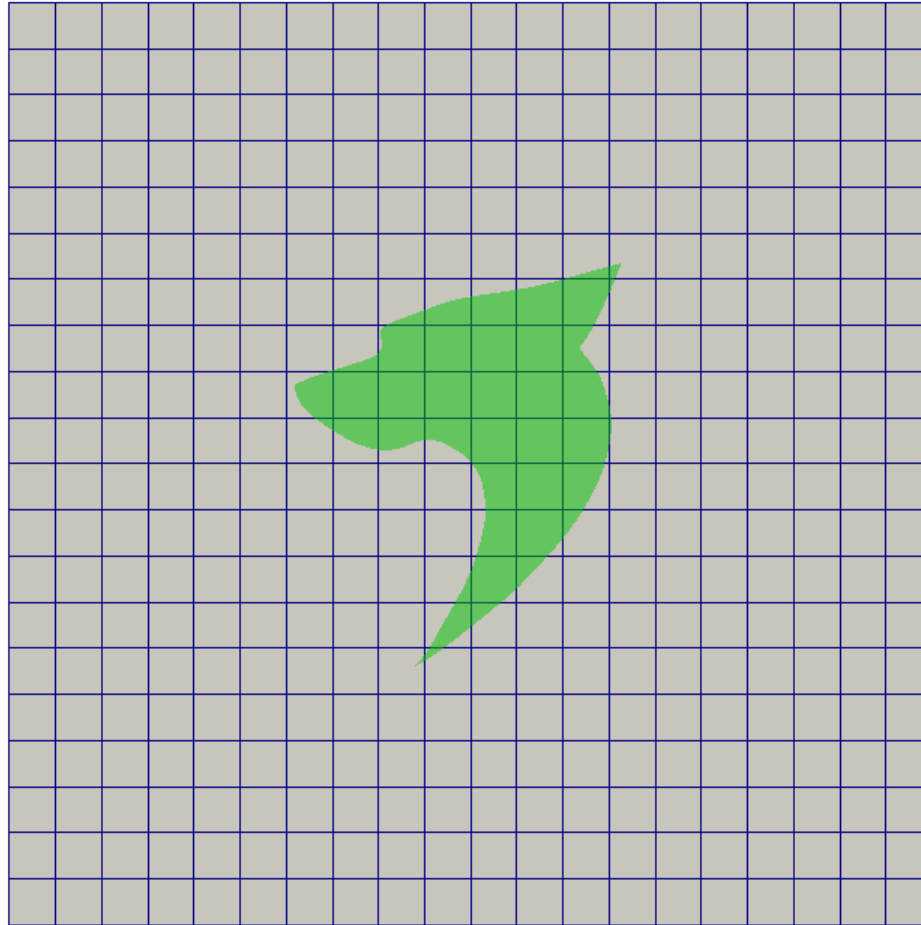
Step 7. Mesh layers

- The mesh output from the snapping stage may be suitable for simulation, although it can produce some irregular cells along boundary surfaces.
- There is an optional stage of the meshing process which introduces boundary layer meshing in selected parts of the mesh.
- This information is supplied by the user in the **addLayersControls** sub-dictionary in the *snappyHexMeshDict* dictionary.
- This is the final step of the mesh generation process using *snappyHexMesh*.
- This is a valid body fitted mesh with boundary layer meshing, that can be used for a simulation.

Mesh generation using snappyHexMesh

snappyHexMesh in action

www.wolfdynamics.com/wiki/shm/ani.gif



Mesh generation using snappyHexMesh

- Let us study the `snappyHexMesh` dictionary in details.
- We are going to work with the case we just saw in action.
- You will find this case in the directory:

M101_wolf

Mesh generation using snappyHexMesh

Let us explore the `snappyHexMeshDict` dictionary.

- Open the dictionary `snappyHexMeshDict` with your favorite text editor (we use gedit).
- The dictionary `snappyHexMeshDict` consists of five main sections:
 - **geometry**
Definition of geometry entities to be used for meshing
 - **castellatedMeshControls**
Definition of feature, surface and volume mesh refinement
 - **snapControls**
Definition of surface mesh snapping and advanced parameters
 - **addLayersControls**
Definition of boundary layer meshing and advanced parameters
 - **meshQualityControls**
Definition of mesh quality metrics

Mesh generation using snappyHexMesh

Let us explore the snappyHexMeshDict dictionary.

```
castellatedMesh true;      //or false
snap true;                //or false
addLayers true;           //or false
```

geometry

```
{
    ...
}
```

castellatedMeshControls

```
{
    ...
}
```

snapControls

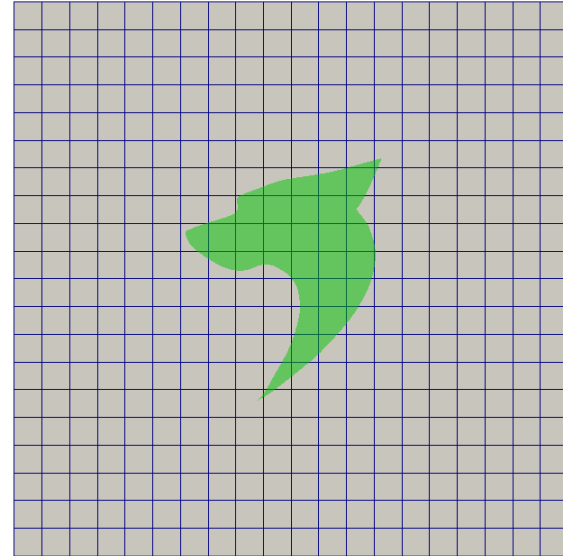
```
{
    ...
}
```

addLayersControls

```
{
    ...
}
```

meshQualityControls

```
{
    ...
}
```



- The `snappyHexMesh` dictionary is made up of five sections, namely: **geometry**, **castellatedMeshControls**, **snapControls**, **addLayersControls** and **meshQualityControls**. Each section controls a step of the meshing process.
- In the first three lines we can turn off and turn on the different meshing steps. For example, if we want to generate a body fitted mesh with no boundary layer we should proceed as follows:

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

Mesh generation using snappyHexMesh

Let us explore the `snappyHexMeshDict` dictionary.

```
castellatedMesh true;      //or false
snap true;                //or false
addLayers true;           //or false
```

```
geometry
```

```
{
    ...
}
```

```
castellatedMeshControls
```

```
{
    ...
}
```

```
snapControls
```

```
{
    ...
}
```

```
addLayersControls
```

```
{
    ...
}
```

```
meshQualityControls
```

```
{
    ...
}
```

May be located in a separated file

- Have in mind that there are more than 70 parameters to control in `snappyHexMeshDict` dictionary.
- Adding the fact that there is no native GUI, it can be quite tricky to control the mesh generation process.
- Nevertheless, `snappyHexMesh` generates really good hexa dominant meshes.
- Hereafter, we will only comment on the most important parameters.
- The parameters that you will find in the `snappyHexMeshDict` dictionaries distributed with the tutorials, in our opinion are robust and will work most of the times.

Mesh generation using snappyHexMesh

Let us explore the snappyHexMeshDict dictionary.

Geometry controls section

```
geometry
{
    wolfExtruded.stl
    {
        type triSurfaceMesh;
        name wolf;
        regions
        {
            wolflocal
            {
                name wolf_wall;
            }
        }
    }
    box
    {
        type searchableBox;
        min (-100.0 -120.0 -50.0 );
        max (100.0 120.0 150.0 );
    }
    sphere
    {
        type searchableSphere;
        centre (120.0 -100.0 50.0 );
        radius 40.0;
    }
}
```

STL file to read

Name of the surface inside snappyHexMesh

Use this option if you have a STL with multiple patches defined

This is the name of the region or surface patch in the STL

User-defined patch name. This is the final name of the patch

Name of geometrical entity

Name of geometrical entity

Note 1

- In this section we read in the STL geometry. Remember, the input geometry is always located in the directory **constant/triSurface**
- We can also define geometrical entities that can be used to refine the mesh, create regions, or generate baffles.
- You can add multiple STL files.
- If you do not give a name to the surface, it will take the name of the STL file.
- The geometrical entities are created inside **snappyHexMesh**.

Note 1:

If you want to know what geometrical entities are available, just misspelled something in the **type** keyword.

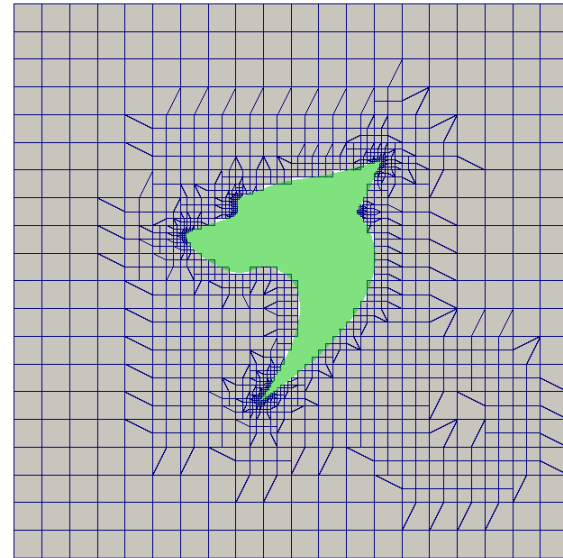
Mesh generation using snappyHexMesh

Let us explore the snappyHexMeshDict dictionary.

castellatedMeshControls

```
{  
    //Refinement parameters  
    maxLocalCells 100000;  
    maxGlobalCells 2000000;  
    nCellsBetweenLevels 1;  
    ...  
    ...  
    //Explicit feature edge refinement  
    features ← Dictionary block  
    (  
        ...  
        ...  
    );  
    //Surface based refinement  
    refinementSurfaces ← Dictionary block  
    {  
        ...  
        ...  
    }  
    //Region-wise refinement  
    refinementRegions ← Dictionary block  
    {  
        ...  
        ...  
    }  
    //Mesh selection  
    locationInMesh (-100.0 0.0 50.0 ); ← Note 1  
}
```

Castellated mesh controls section



- In the **castellatedMeshControls** section, we define the global refinement parameters, explicit feature edge refinement, surface based refinement, region-wise refinement and the material point.
- In this step, we are generating the castellated mesh.

Note 1:

The material point indicates where we want to create the mesh, that is, inside or outside the body to be meshed.

Mesh generation using snappyHexMesh

Let us explore the snappyHexMeshDict dictionary.

castellatedMeshControls

{

// Refinement parameters

```
maxLocalCells 100000;  
maxGlobalCells 2000000;  
minRefinementCells 0;  
maxLoadUnbalance 0.10;  
nCellsBetweenLevels 1;
```

← Note 1

//Local curvature and //feature angle refinement

```
resolveFeatureAngle 30;
```

← Note 2

```
planarAngle 30;
```

```
allowFreeStandingZoneFaces true;
```

//Explicit feature edge refinement

```
features ← Dictionary block
```

```
{
```

```
{
```

```
file "wolfExtruded.eMesh";  
level 2;
```

← Note 3

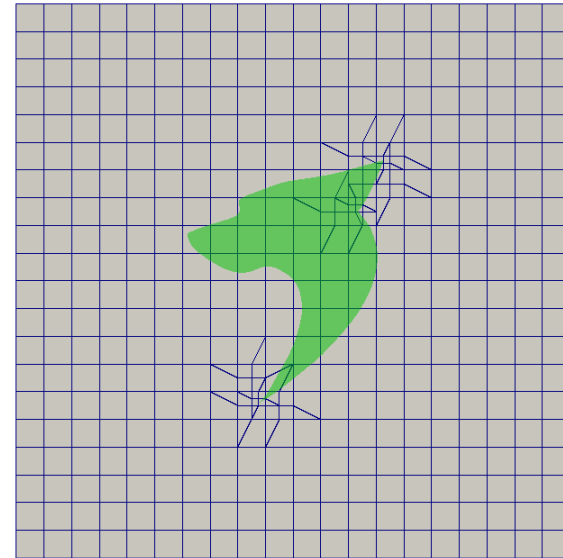
```
}
```

```
};
```

```
...  
...  
...
```

}

Castellated mesh controls section



Note 1:

This parameter controls the transition between cell refinement levels.

Note 2:

This parameter controls the local curvature refinement. The higher the value, the less features it captures. For example if you use 100, it will not add refinement in high curvature areas. It also controls edge feature snapping, high values will not resolve sharp angles in surface intersections.

Note 3:

This file is automatically created when you use the utility `surfaceFeatureExtract`. The file is located in the directory `constant/triSurface`

Mesh generation using snappyHexMesh

Let us explore the snappyHexMeshDict dictionary.

```
castellatedMeshControls
```

```
{
```

```
...  
...  
...
```

```
//Surface based refinement
```

```
refinementSurfaces
```

← Dictionary block

```
{
```

```
//wolf was defined in the geometry section
```

```
wolf
```

```
{
```

```
level (1 1);
```

```
//Global refinement
```

```
regions
```

```
{
```

```
wolflocal
```

```
{
```

```
level (2 4);
```

```
patchInfo
```

```
{
```

```
type wall;
```

```
}
```

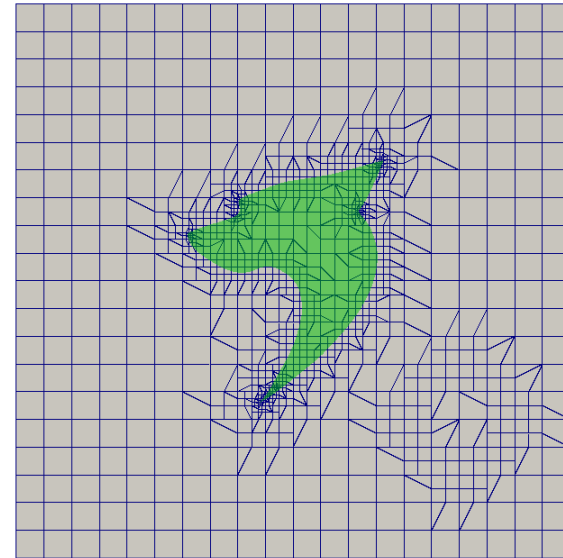
```
}
```

```
}
```

```
}
```

```
...  
...  
}
```

Castellated mesh controls section



Note 1:

The surface **wolf** was defined in the geometry section.

Note 2:

The region **wolflocal** was defined in the geometry section.

Note 3:

Named region in the STL file. This refinement is local. To use the surface refinement in the regions, the local regions must exist in STL file. We created a pointer to this region in the **geometry** section.

Note 4:

You can only define patches of type wall or patch.

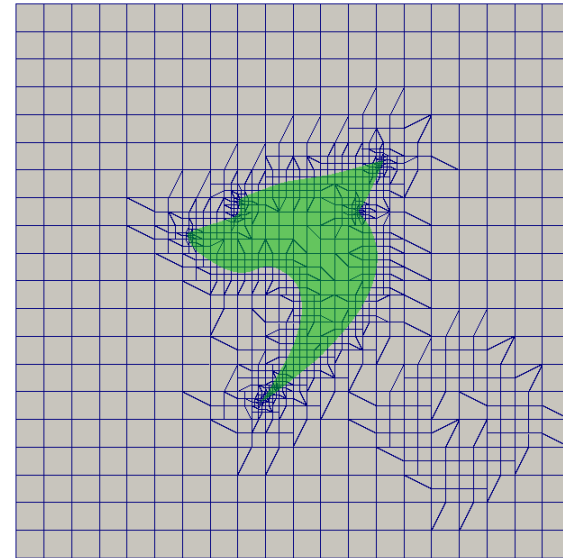
Mesh generation using snappyHexMesh

Let us explore the snappyHexMeshDict dictionary.

castellatedMeshControls

```
{  
  
    //Surface based refinement  
    refinementSurfaces ← Dictionary block  
    {  
  
        ...  
  
        ...  
  
        //This surface or geometrical entity  
        //was defined in geometry section  
        sphere ← Note 1  
        {  
  
            level (1 1);  
  
            faceZone face_inner;  
            cellZone cell_inner;  
  
            cellZoneInside inside;  
  
            //faceType internal;  
  
        }  
  
        ...  
  
    }  
}
```

Castellated mesh controls section



Note 1:

Optional specification of what to do with **faceZone** faces:

internal: keep them as internal faces (default)

baffle: create baffles from them. This gives more freedom in mesh motion

boundary: create free-standing boundary faces (baffles but without the shared points)

e.g., **faceType** internal;

Mesh generation using snappyHexMesh

Let us explore the snappyHexMeshDict dictionary.

```
castellatedMeshControls
```

```
{
```

```
...  
...  
...
```

```
//Region-wise refinement
```

```
refinementRegions ← Dictionary block
```

```
{
```

```
//This region or geometrical entity  
//was defined in the geometry section
```

```
box ← Note 1
```

```
{
```

```
mode inside;  
levels (( 1 1 ));
```

```
}
```

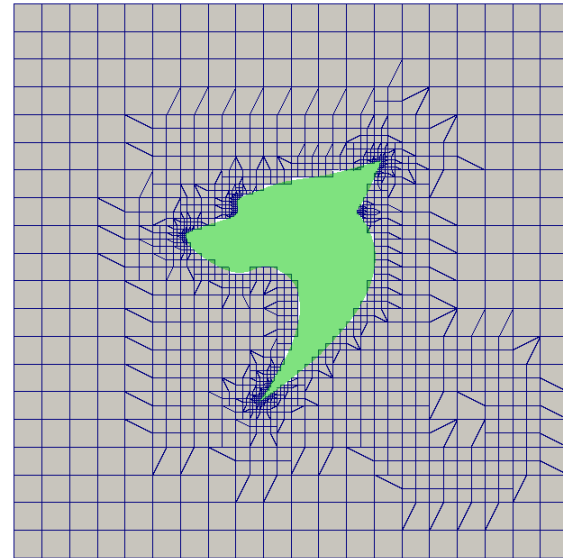
```
}
```

```
//Mesh selection
```

```
locationInMesh (-100.0 0.0 50.0 );
```

```
}
```

Castellated mesh controls section



Note 1:

This region or geometrical entity was created in the **geometry** section.

At this point we have a valid mesh (cartesian)

Mesh generation using snappyHexMesh

Let us explore the snappyHexMeshDict dictionary.

```
castellatedMeshControls
```

```
{
```

```
...  
...  
...
```

```
//Region-wise refinement
```

```
refinementRegions
```

← Dictionary block

```
{
```

```
//This region or geometrical entity  
//was defined in the geometry section
```

```
box
```

```
{
```

```
mode inside;  
levels (( 1 1 ));
```

```
}
```

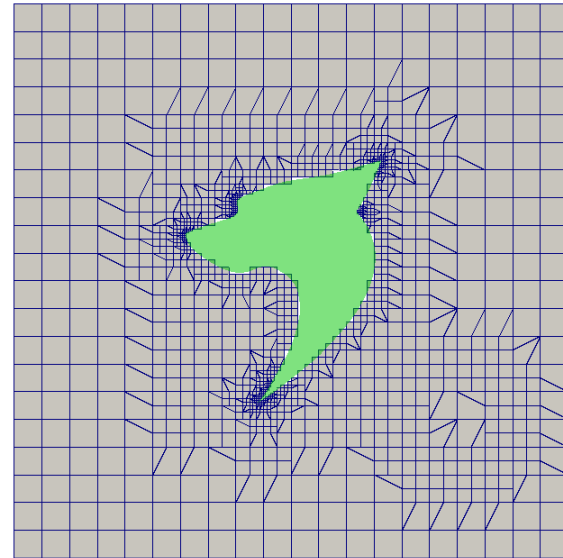
```
}
```

```
//Mesh selection
```

```
locationInMesh (-100.0 0.0 50.0 );
```

```
}
```

Castellated mesh controls section



This point defines where do you want the mesh.
Can be internal mesh or external mesh.

- If the point is inside the STL it is an internal mesh.
- If the point is inside the background mesh and outside the STL it is an external mesh.

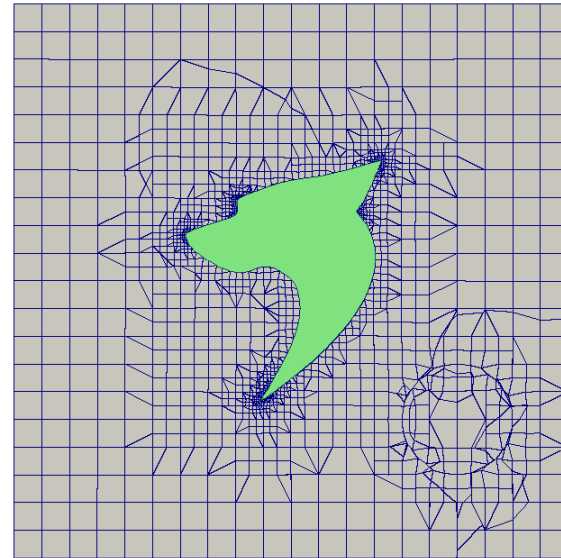
Mesh generation using snappyHexMesh

Let us explore the snappyHexMeshDict dictionary.

snapControls

```
{  
  
    //Number of patch smoothing iterations  
    //before finding correspondence to surface  
    nSmoothPatch 3;  
  
    tolerance 2.0;  
  
    //- Number of mesh displacement relaxation  
    //iterations.  
    nSolverIter 30; ← Note 1  
  
    //- Maximum number of snapping relaxation  
    //iterations. Should stop before upon  
    //reaching a correct mesh.  
    nRelaxIter 5; ← Note 2  
  
    // Feature snapping  
  
    //Number of feature edge snapping iterations.  
    nFeatureSnapIter 10; ← Note 3  
  
    //Detect (geometric only) features by  
    //sampling the surface (default=false).  
    implicitFeatureSnap false;  
  
    // Use castellatedMeshControls::features  
    // (default = true)  
    explicitFeatureSnap true;  
  
    multiRegionFeatureSnap false;  
  
}
```

Snap mesh controls section



Note 1:

The higher the value the better the body fitted mesh. The default value is 30. If you are having problems with the mesh quality (related to the snapping step), try to increase this value to 300. Have in mind that this will increase the meshing time.

Note 2:

Increase this value to improve the quality of the body fitted mesh.

Note 3:

Increase this value to improve the quality of the edge features.

- In this step, we are generating the body fitted mesh.

Mesh generation using snappyHexMesh

Let us explore the snappyHexMeshDict dictionary.

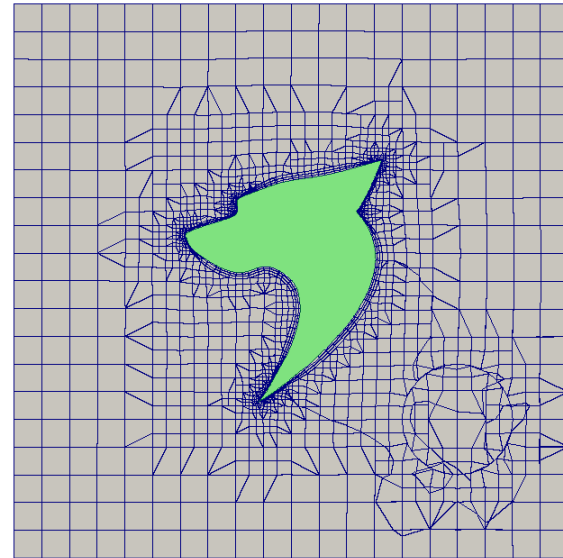
addLayersControls

```
{
    //Global parameters
    relativeSizes true;
    expansionRatio 1.2;
    finalLayerThickness 0.5;
    minThickness 0.1;

    layers ←———— Note 1
    {
        wolf_wall ←———— Note 2
        {
            nSurfaceLayers 3;
            //Local parameters
            //expansionRatio 1.3;
            //finalLayerThickness 0.3;
            //minThickness 0.1;
        }
    }

    // Advanced settings
    nGrow 0;
    featureAngle 130; ←———— Note 3
    maxFaceThicknessRatio 0.5;
    nSmoothSurfaceNormals 1;
    nSmoothThickness 10;
    minMedianAxisAngle 90;
    maxThicknessToMedialRatio 0.3;
    nSmoothNormals 3;
    slipFeatureAngle 30;
    nRelaxIter 5;
    nBufferCellsNoExtrude 0;
    nLayerIter 50;
    nRelaxedIter 20;
}
```

Boundary layer mesh controls section



Note 1:

In this section we select the patches where we want to add the layers. We can add multiple patches (if they exist).

Note 2:

This patch was created in the **geometry** section.

Note 3:

Specification of feature angle above which layers are collapsed automatically.

- In this step, we are generating the boundary layer mesh.

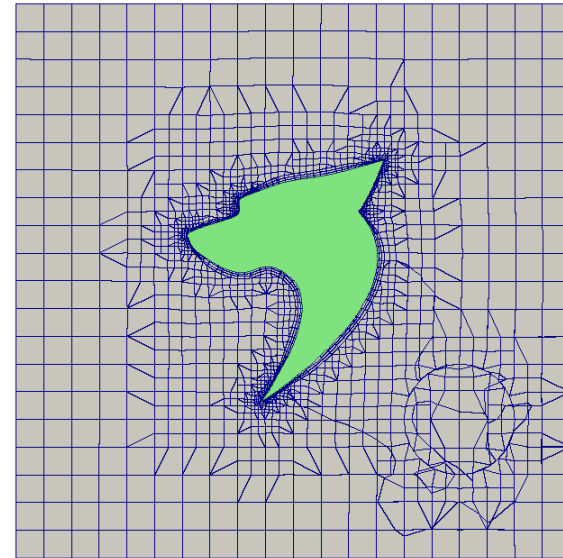
Mesh generation using snappyHexMesh

Let us explore the snappyHexMeshDict dictionary.

meshQualityControls

```
{  
    maxNonOrtho 75; ← Note 1  
    maxBoundarySkewness 20;  
    maxInternalSkewness 4; ← Note 2  
    maxConcave 80;  
    minVol 1.00E-13;  
    minTetQuality 1e15;  
    //minTetQuality -1e30;  
    minArea -1;  
    minTwist 0.02;  
    minDeterminant 0.001;  
    minFaceWeight 0.05;  
    minVolRatio 0.01;  
    minTriangleTwist -1;  
    minFlatness 0.5;  
    nSmoothScale 4;  
    errorReduction 0.75;  
}
```

Mesh quality controls section



Note 1:

Maximum non-orthogonality angle.

Note 2:

Maximum skewness angle.

- During the mesh generation process, the mesh quality is continuously monitored.
- The mesher `snappyHexMesh` will try to generate a mesh using the mesh quality parameters defined by the user.
- If a mesh motion or topology change introduces a poor quality cell or face the motion or topology change is undone to revert the mesh back to a previously valid error free state.

Mesh generation using snappyHexMesh

Let us explore the snappyHexMeshDict dictionary.

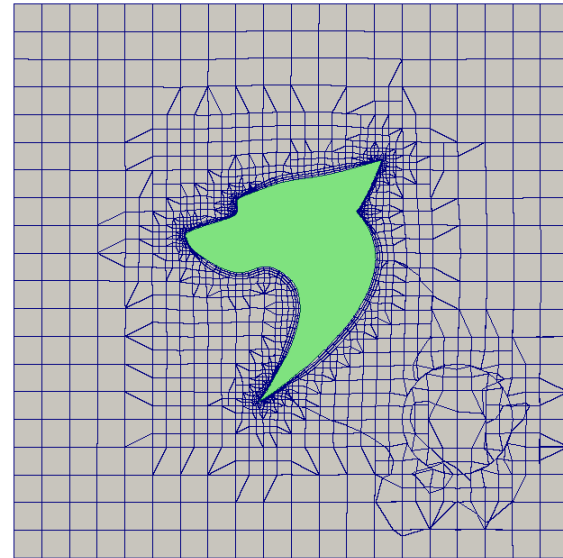
debugFlags

```
(  
    // write intermediate meshes  
    mesh  
  
    // write current mesh intersections as .obj files  
    intersections  
  
    // write information about explicit feature edge  
    // refinement  
    featureSeeds  
  
    // write attraction as .obj files  
    attraction  
  
    // write information about layers  
    layerInfo  
);
```

writeFlags

```
(  
    // write volScalarField with cellLevel for  
    // postprocessing  
    scalarLevels  
  
    // write cellSets, faceSets of faces in layer  
    layerSets  
  
    // write volScalarField for layer coverage  
    layerFields  
);
```

Mesh debug and write controls sections



- At the end of the dictionary you will find the sections: **debugFlags** and **writeFlags**
- By default they are commented. If you uncomment them you will enable debug information.
- **debugFlags** and **writeFlags** will produce a lot of outputs that you can use to post process and troubleshoot the different steps of the meshing process.

Mesh generation using snappyHexMesh

Let us generate the mesh of the wolf dynamics logo.

- This tutorial is located in the directory:
 - `M101_wolf`
- In this case we are going to generate a body fitted mesh with boundary layer. This is an external mesh.
- Before generating the mesh take a look at the dictionaries and files that will be used.
- These are the dictionaries and files that will be used.
 - `system/snappyHexMeshDict`
 - `system/surfaceFeatureExtractDict`
 - `system/meshQualityDict`
 - `system/blockMeshDict`
 - `constant/triSurface/wolfExtruded.stl`
 - `constant/triSurface/wolfExtruded.eMesh`
- The file `wolfExtruded.eMesh` is generated after using the utility `surfaceFeatureExtract`, which reads the dictionary `surfaceFeatureExtractDict`.

Mesh generation using snappyHexMesh

Let us generate the mesh of the wolf dynamics logo.

- To generate the mesh, in the terminal window type:
 1. `$> foamCleanTutorials`
 2. `$> blockMesh`
 3. `$> surfaceFeatureExtract`
 4. `$> snappyHexMesh`
 5. `$> checkMesh -latestTime`
- To visualize the mesh, in the terminal window type:
 - `$> paraFoam`
- Remember to use the VCR controls in paraView/paraFoam to visualize the mesh intermediate steps.

Mesh generation using snappyHexMesh

Let us generate the mesh of the wolf dynamics logo.

- In the case directory you will find the time folders 1, 2, and 3, which contain the castellated mesh, snapped mesh and boundary layer mesh respectively. In this case, `snappyHexMesh` automatically saved the intermediate steps.
- Before running the simulation, remember to transfer the solution from the latest mesh to the directory **constant/polyMesh**, in the terminal type:

```
1. $> cp 3/polyMesh/* constant/polyMesh
2. $> rm -rf 1
3. $> rm -rf 2
4. $> rm -rf 3
5. $> checkMesh -latestTime
```

Mesh generation using snappyHexMesh

Let us generate the mesh of the wolf dynamics logo.

- If you want to avoid the additional steps of transferring the final mesh to the directory **constant/polyMesh** by not saving the intermediate steps, you can proceed as follows:
 - `$> snappyHexMesh -overwrite`
- When you proceed in this way, `snappyHexMesh` automatically saves the final mesh in the directory **constant/polyMesh**.
- Have in mind that you will not be able to visualize the intermediate steps.
- Also, you will not be able to restart the meshing process from a saved state (castellated or snapped mesh).
- Unless it is strictly necessary, from this point on we will not save the intermediate steps.



Mesh generation using snappyHexMesh

Let us generate the mesh of the wolf dynamics logo.

- At this point, we have a valid mesh to run a simulation.
- Have in mind that before running the simulation you will need to set the boundary and initial conditions in the directory 0.
- The **name** and **base type** information of the boundary patches is saved in the file *constant/polyMesh/boundary*.
- Remember, the **base type** (patch type defined in the file *constant/polyMesh/boundary*) and the **primitive type** of the boundary conditions (patch type defined in the fields dictionary in the directory 0), must be compatible.
- You also need to use the same naming convention. That is, the name of the patches defined in the file *constant/polyMesh/boundary* and the name of the patches defined in the files inside the directory 0, must be the same.

Mesh generation using snappyHexMesh

Let us generate the mesh of the wolf dynamics logo.

The *constant/polyMesh/boundary* dictionary

- First at all, this file is automatically generated after you create the mesh or you convert it from a third party format.
- In this file, the geometrical information related to the **base type** patch of each boundary of the domain is specified.
- The **base type** boundary condition is the actual surface patch where we are going to apply a **primitive type** boundary condition (or numerical boundary condition).
- The **primitive type** boundary condition assign a field value to the surface patch (**base type**).
- You define the **numerical type** patch (or the value of the boundary condition), in the directory 0 or time directories.
- The **name** and **base type** of the patches was defined in the dictionaries *blockMeshDict* and *snappyHexMeshDict*.
- You can change the **name** if you do not like it. Do not use strange symbols or white spaces.
- You can also change the **base type**. For instance, you can change the type of the patch **maxY** from **wall** to **patch**.

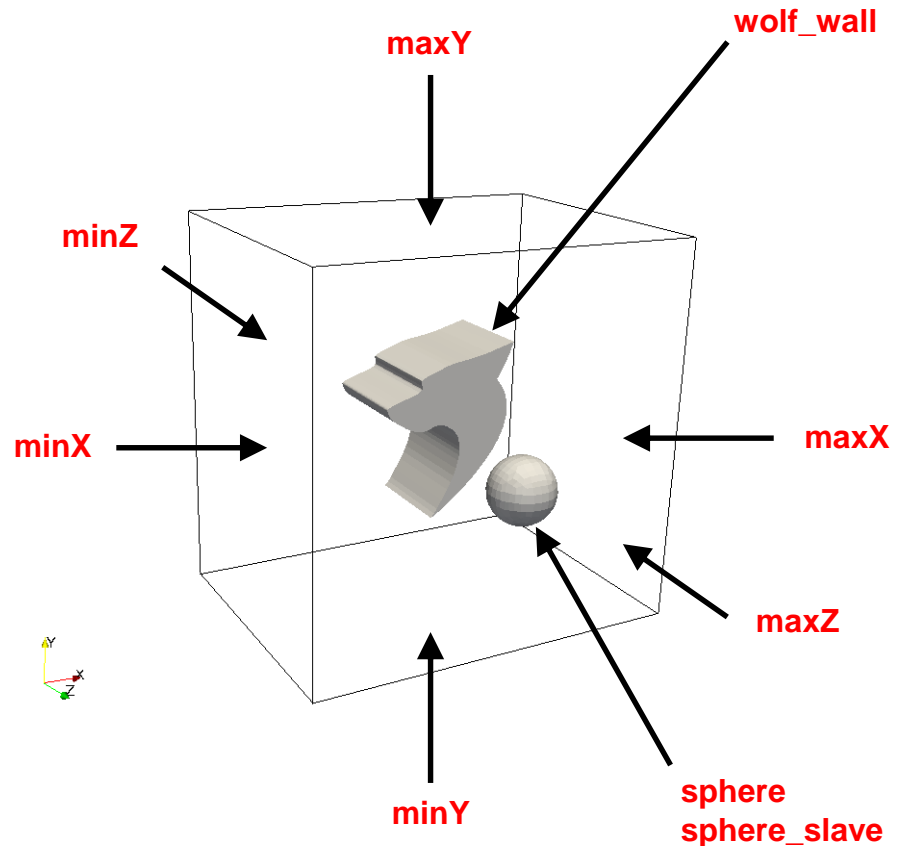
Mesh generation using snappyHexMesh

Let us generate the mesh of the wolf dynamics logo.

```
18 9  
19 (  
20   minX  
21   {  
22     type      wall;  
23     inGroups   1(wall);  
24     nFaces     400;  
25     startFace  466399;  
26   }  
27   maxX  
28   {  
29     type      wall;  
30     inGroups   1(wall);  
31     nFaces     400;  
32     startFace  466799;  
33   }  
34   minY  
35   {  
36     type      empty;  
37     inGroups   1(wall);  
38     nFaces     400;  
39     startFace  467199;  
40   }  
41   maxY  
42   {  
43     type      wall;  
44     inGroups   1(wall);  
45     nFaces     400;  
46     startFace  467599;  
47   }  
48   minZ  
49   {  
50     type      wall;  
51     inGroups   1(wall);  
52     nFaces     400;  
53     startFace  467999;  
54   }
```

Number of surface patches

In the list below there must be 9 patches definition.



Mesh generation using snappyHexMesh

Let us generate the mesh of the wolf dynamics logo.

```
18 9
19 (
20   minX
21   {
22     type      wall;
23     inGroups   1 (wall);
24     nFaces     400;
25     startFace  466399;
26   }
27   maxX
28   {
29     type      wall;
30     inGroups   1 (wall);
31     nFaces     400;
32     startFace  466799;
33   }
34   minY
35   {
36     type      empty;
37     inGroups   1 (wall);
38     nFaces     400;
39     startFace  467199;
40   }
41   maxY
42   {
43     type      wall;
44     inGroups   1 (wall);
45     nFaces     400;
46     startFace  467599;
47   }
48   minZ
49   {
50     type      wall;
51     inGroups   1 (wall);
52     nFaces     400;
53     startFace  467999;
54   }
```


Annotations in the image:

- Name**: points to the patch name (e.g., `minX`).
- Type**: points to the patch type (e.g., `wall`).
- nFaces**: points to the `nFaces` keyword.
- startFace**: points to the `startFace` keyword.

Name and type of the surface patches

- The **name** and **base type** of the patch is given by the user.
- In this case the **name** and **base type** was assigned in the dictionaries *blockMeshDict* and *snappyHexMeshDict*.
- You can change the **name** if you do not like it. Do not use strange symbols or white spaces.
- You can also change the **base type**. For instance, you can change the type of the patch **maxY** from **wall** to **patch**.

nFaces and startFace keywords

- Unless you know what you are doing, **you do not need to change this information.** 
- Basically, this is telling you the starting face and ending face of the patch.
- This information is created automatically when generating the mesh or converting the mesh.

Mesh generation using snappyHexMesh


Let us generate the mesh of the wolf dynamics logo.

```
55     maxZ
56     {
57         type            wall;
58         inGroups        1(wall);
59         nFaces          400;
60         startFace       466399;
61     }
62     wolf_wall ← Name
63     {
64         type            wall; ← Type
65         inGroups        1(wall);
66         nFaces          400;
67         startFace       466799; ← nFaces startFace
68     }
69     sphere
70     {
71         type            empty;
72         inGroups        1(wall);
73         nFaces          400;
74         startFace       467199;
75     }
76     sphere_slave
77     {
78         type            wall;
79         inGroups        1(wall);
80         nFaces          400;
81         startFace       467599;
82     }
83 }
```

Name and type of the surface patches

- The **name** and **base type** of the patch is given by the user.
- In this case the **name** and **base type** was assigned in the dictionaries *blockMeshDict* and *snappyHexMeshDict*.
- You can change the **name** if you do not like it. Do not use strange symbols or white spaces.
- You can also change the **base type**. For instance, you can change the type of the patch **maxY** from **wall** to **patch**.

nFaces and startFace keywords

- Unless you know what you are doing, **you do not need to change this information.** 
- Basically, this is telling you the starting face and ending face of the patch.
- This information is created automatically when generating the mesh or converting the mesh.

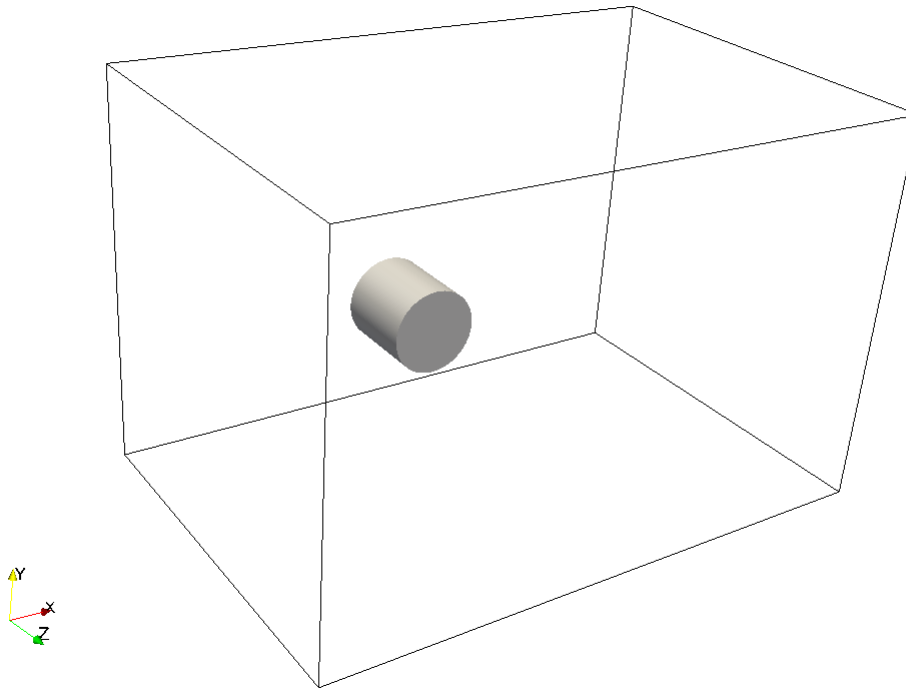
Mesh generation using snappyHexMesh

Cleaning the case directory

- When generating the mesh using OpenFOAM®, it is extremely important to start from a clean case directory.
- To clean all the case directory, in the terminal type:
 - `$> foamCleanTutorials`
- To only erase the mesh information, in the terminal type:
 - `$> foamCleanPolyMesh`
- If you are planning to start the meshing from a previous saved state, you do not need to clean the case directory.
- Before proceeding to compute the solution, remember to always check the quality of the mesh.

snappyHexMesh guided tutorials

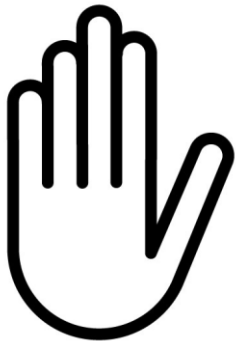
- Our first case will be a mesh around a cylinder.
- This is a simple geometry, but we will use it to study all the meshing steps and introduce a few advanced features.
- This case is located in the directory **M1cy1**



snappyHexMesh guided tutorials

- Meshing with snappyHexMesh.
- Meshing case 1. 3D Cylinder (external mesh), with feature edge refinement.

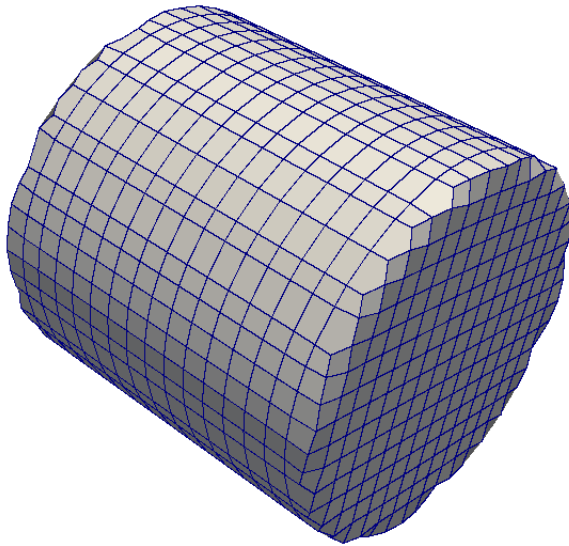
M1cy1/C1



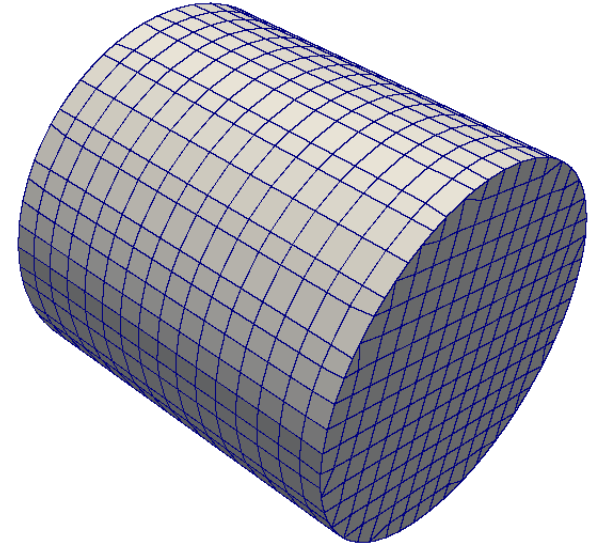
- From this point on, please follow me.
- We are all going to work at the same pace.
- Remember, \$PTOFC is pointing to the path where you unpacked the tutorials.

snappyHexMesh guided tutorials

3D Cylinder with edge refinement.



No edge refinement

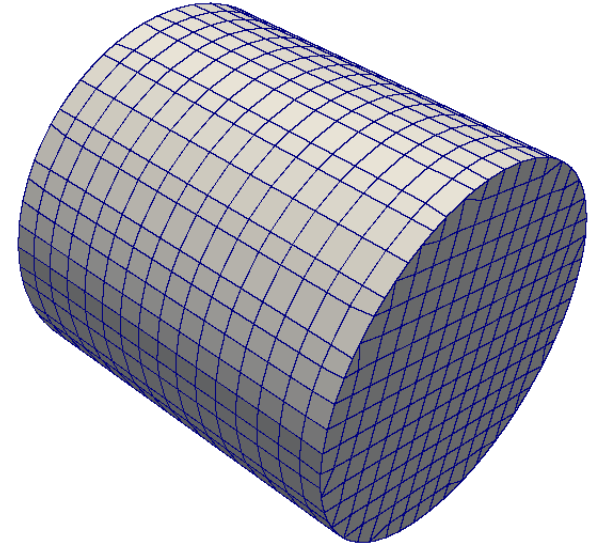
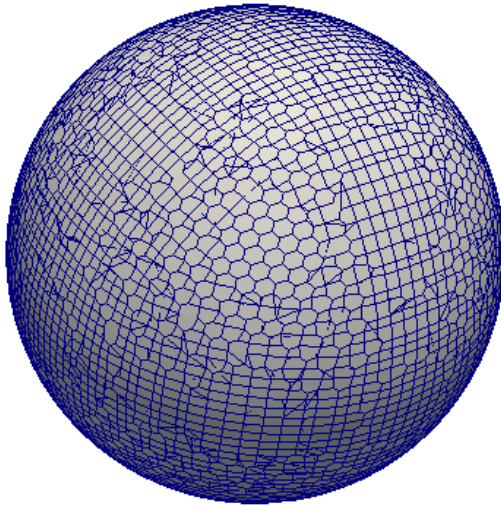


Edge refinement

- If the geometry has sharp angles and you want to resolve those edges, you should use edge refinement.
- In this case and as we are interested in resolving the sharp angles, the mesh with no edge refinement is not acceptable.

snappyHexMesh guided tutorials

3D Cylinder with edge refinement.



- We use edge refinement when we have sharp angles.
- In the left figure there is no need to use edge refinement
- In the right figure (and depending of what you want to do), you will need to use edge refinement.

snappyHexMesh guided tutorials

3D Cylinder with edge refinement.

- How do we control curvature refinement and enable edge refinement?
- In the file *snappyHexMeshDict*, look for the following entry:

```
castellatedMeshControls
```

```
{
```

```
...
```

```
...
```

```
...
```

```
//Local curvature and  
//feature angle refinement  
resolveFeatureAngle 30;
```

← To control curvature refinement

```
...
```

```
...
```

```
...
```

```
//Explicit feature edge refinement  
features
```

```
{
```

```
{
```

```
file "surfacemesh.eMesh";  
level 0;
```

← To enable and
control edge
refinement level

```
}
```

```
);
```

```
...
```

```
...
```

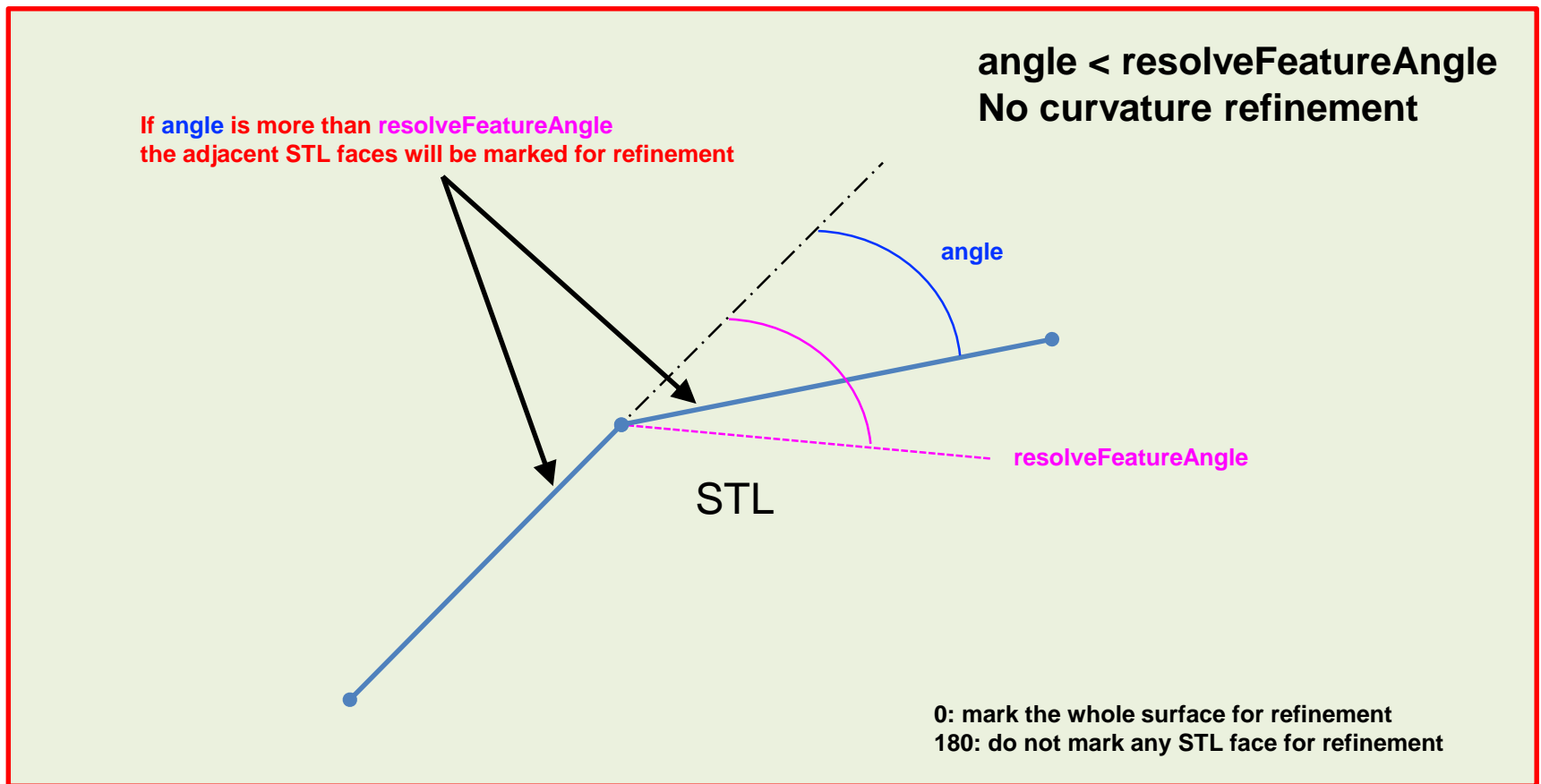
```
...
```

```
}
```

snappyHexMesh guided tutorials

3D Cylinder with edge refinement.

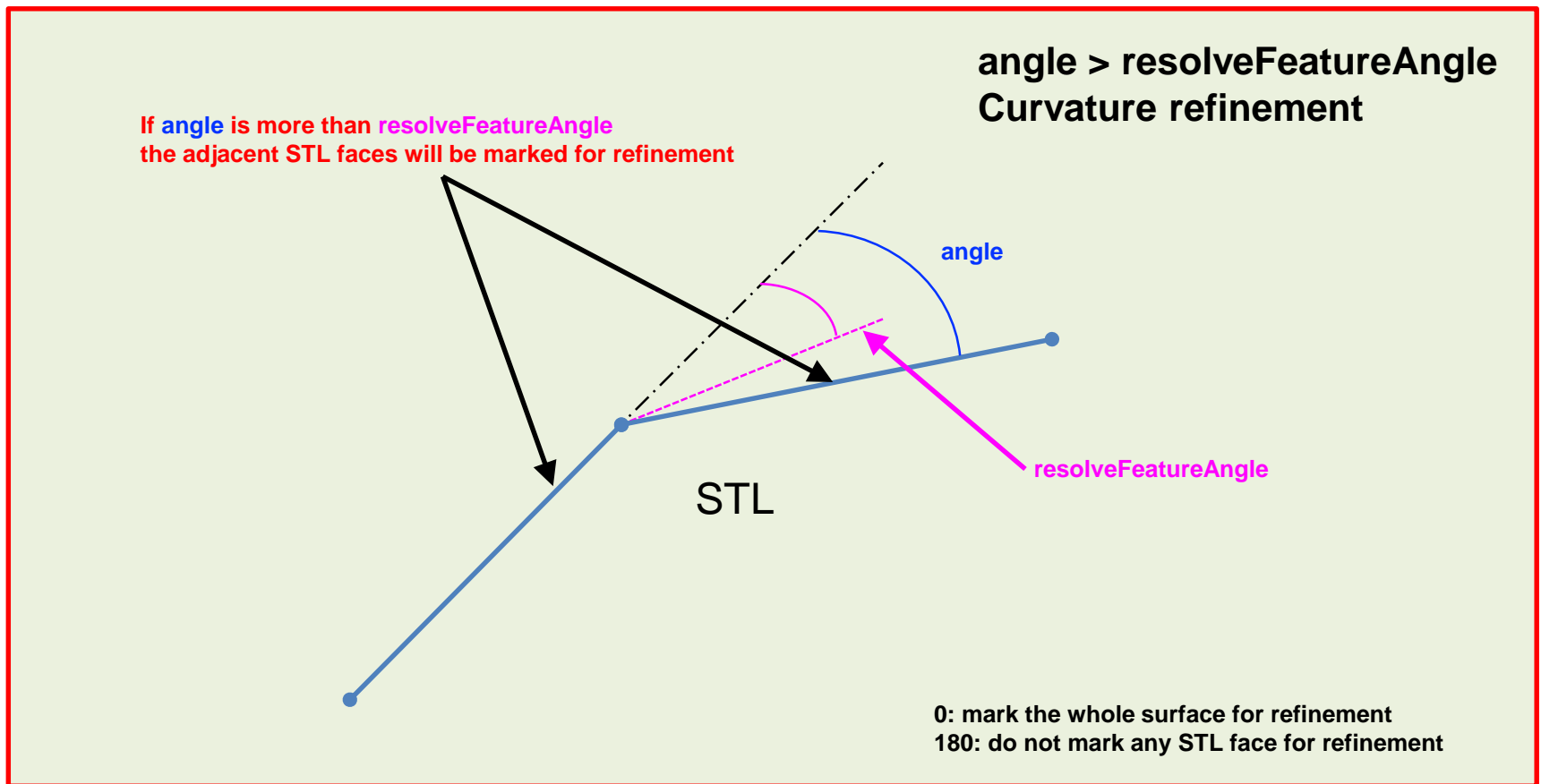
How **resolveFeatureAngle** works?



snappyHexMesh guided tutorials

3D Cylinder with edge refinement.

How **resolveFeatureAngle** works?



snappyHexMesh guided tutorials

3D Cylinder with edge refinement.

- How do we control surface refinement?
- In the file *snappyHexMeshDict*, look for the following entry:

```
castellatedMeshControls
{
    ...
    ...
    //Surface based refinement
    refinementSurfaces
    {
        banana_stlSurface
        {
            level (2 4);
        }
    }
    ...
    ...
    ...
}
```

To control surface refinement.
The first digit controls the global
surface refinement level and the second
digit controls the curvature refinement
level, according to the angle set in the
entry `resolveFeatureAngle`

snappyHexMesh guided tutorials

3D Cylinder with edge refinement.

- How do we create refinement regions?
- In the file *snappyHexMeshDict*, look for the following entry:

```
geometry
```

```
{
```

```
...
```

```
...
```

```
...
```

```
refinementBox
```

```
{
```

```
type searchableBox;
```

```
min ( -2 -2 -2);
```

```
max ( 2 2 2);
```

```
}
```

```
...
```

```
...
```

```
...
```

```
};
```

← Name of refinement region

← Geometrical entity type.
This is the zone where we
want to apply the refinement

← Dimensions of geometrical entity

snappyHexMesh guided tutorials

3D Cylinder with edge refinement.

- How do we create refinement regions?
- In the file *snappyHexMeshDict*, look for the following entry:

```
castellatedMeshControls
```

```
{
```

```
...
```

```
...
```

```
...
```

```
refinementRegions
```

```
{
```

```
refinementBox
```

```
{
```

```
mode inside;
```

```
levels ((1e15 1));
```

```
}
```

```
}
```

```
...
```

```
...
```

```
...
```

```
}
```

Name of the region
created in the geometry section

Type of refinement (inside,
outside, or distance mode)

Refinement level

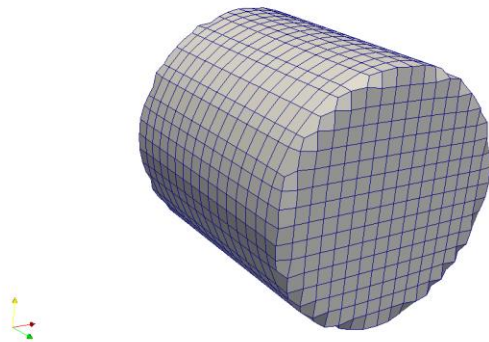
Distance from the surface

A large value covers the whole region

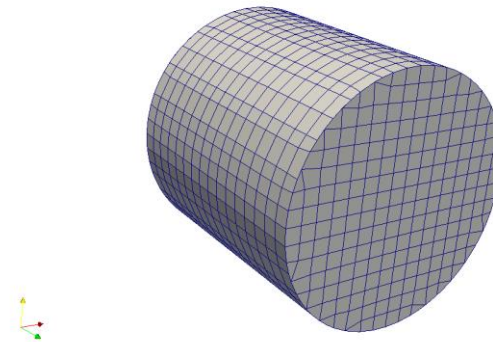
snappyHexMesh guided tutorials

3D Cylinder with edge refinement.

Effect of various parameters on edge capturing and surface refinement



Explicit feature edge refinement level 0
resolveFeatureAngle 110
Surface based refinement level (2 2)



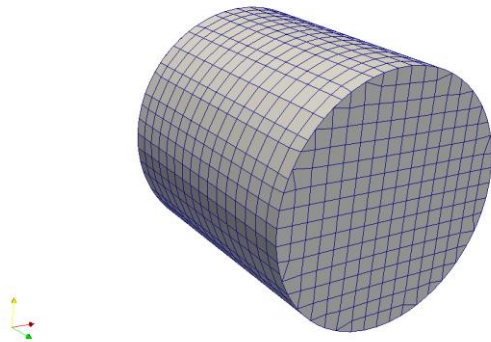
Explicit feature edge refinement level 0
resolveFeatureAngle 60
Surface based refinement level (2 2)

- To control edges capturing you can decrease the value of **resolveFeatureAngle**.
- Be careful, this parameter also controls curvature refinement, so if you choose a low value you also will be adding a lot of refinement on the surface.

snappyHexMesh guided tutorials

3D Cylinder with edge refinement.

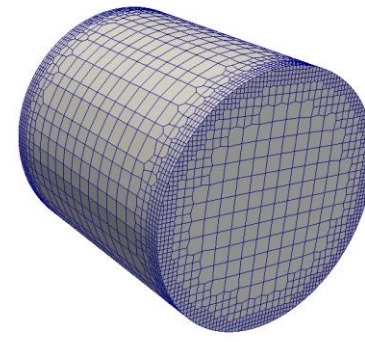
Effect of various parameters on edge capturing and surface refinement



Explicit feature edge refinement level 0

resolveFeatureAngle 60

Surface based refinement level (2 2)



Explicit feature edge refinement level 4

resolveFeatureAngle 60

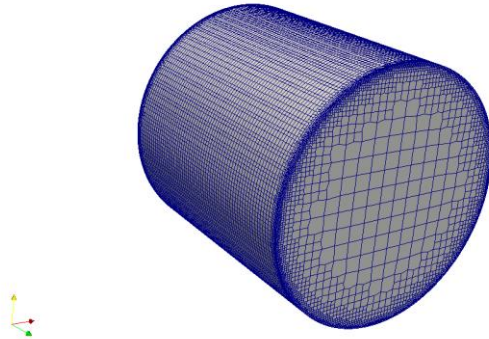
Surface based refinement level (2 2)

- To control edges refinement level, you can change the value of the explicit feature edge refinement level.

snappyHexMesh guided tutorials

3D Cylinder with edge refinement.

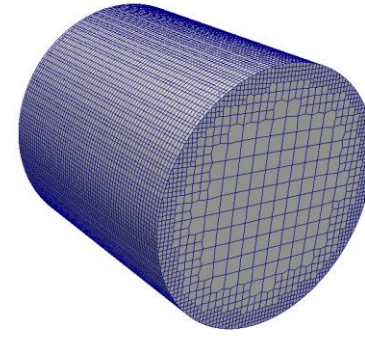
Effect of various parameters on edge capturing and surface refinement



Explicit feature edge refinement level 6

resolveFeatureAngle 5

Surface based refinement level (2 4)



Explicit feature edge refinement level 0

resolveFeatureAngle 5

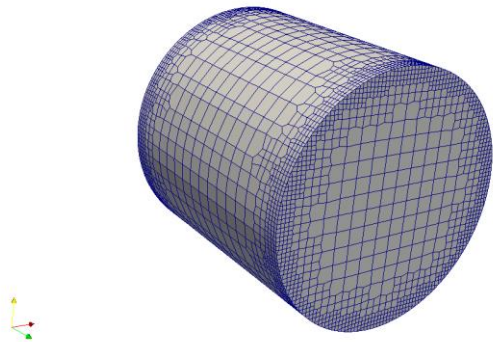
Surface based refinement level (2 4)

- To control edges refinement level, you can change the value of the explicit feature edge refinement level.

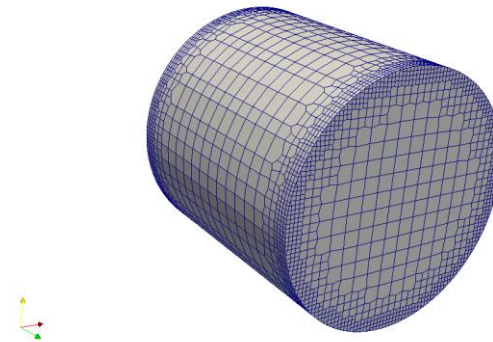
snappyHexMesh guided tutorials

3D Cylinder with edge refinement.

Effect of various parameters on edge capturing and surface refinement



Explicit feature edge refinement level **0**
resolveFeatureAngle 60
Surface based refinement level (2 4)



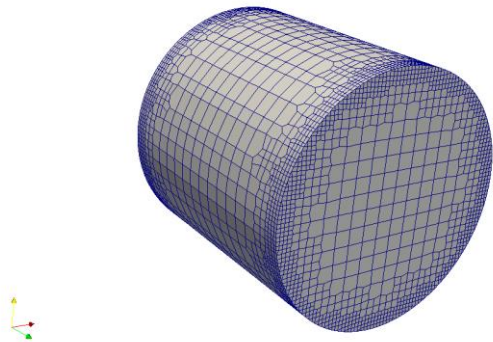
Explicit feature edge refinement level **4**
resolveFeatureAngle 60
Surface based refinement level (2 2)

- To control surface refinement level, you can change the value of the surface based refinement level.
- The first digit controls the global surface refinement level and the second digit controls the curvature refinement level.

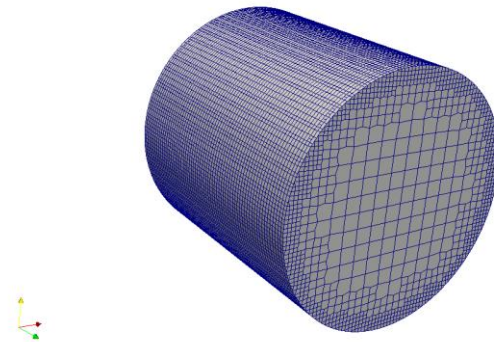
snappyHexMesh guided tutorials

3D Cylinder with edge refinement.

Effect of various parameters on edge capturing and surface refinement



Explicit feature edge refinement level 0
resolveFeatureAngle 60
Surface based refinement level (2 4)



Explicit feature edge refinement level 0
resolveFeatureAngle 5
Surface based refinement level (2 4)

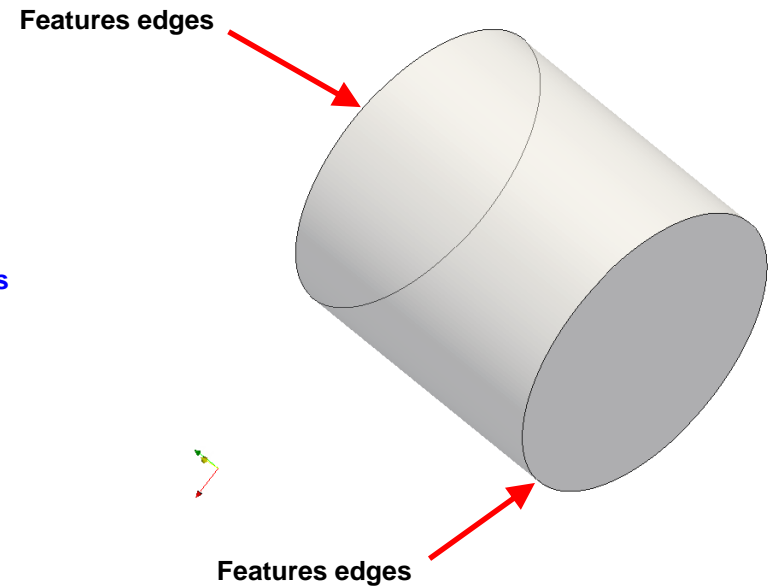
- To control surface refinement due to curvature together with control based surface refinement level, you can change the value of **resolveFeatureAngle**, and surface based refinement level

snappyHexMesh guided tutorials

3D Cylinder with edge refinement.

- Let us explore the dictionary *surfaceFeatureExtractDict* used by the utility *surfaceFeatureExtract*.
- This utility will extract surface features (sharp angles) according to an angle criterion (**includedAngle**).

```
surfacemesh.stl ← Name of the STL.  
{                               The STL file is located  
    extractionMethod    extractFromSurface;  
    extractFromSurfaceCoeffs  
    {  
        includedAngle    150; ← Angle criterion  
    }                    to extract features  
    subsetFeatures  
    {  
        nonManifoldEdges    yes;  
        openEdges           yes;  
    }  
    writeObj            yes; ← If you want to save  
                           the .obj files  
}
```

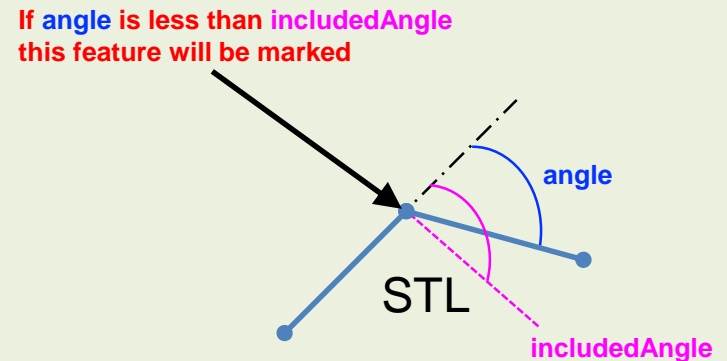


snappyHexMesh guided tutorials

3D Cylinder with edge refinement.

- Let us explore the dictionary *surfaceFeatureExtractDict* used by the utility *surfaceFeatureExtract*.
- This utility will extract surface features (sharp angles) according to an angle criterion (**includedAngle**).

```
surfacemesh.stl ← Name of the STL.  
{                               The STL file is located  
                                in constant/triSurface  
  
    extractionMethod    extractFromSurface;  
  
    extractFromSurfaceCoeffs  
    {  
        includedAngle    150; ← Angle criterion  
                                to extract features  
    }  
  
    subsetFeatures  
    {  
        nonManifoldEdges    yes;  
        openEdges           yes;  
    }  
  
    writeObj    yes; ← If you want to save  
                  the .obj files  
}
```



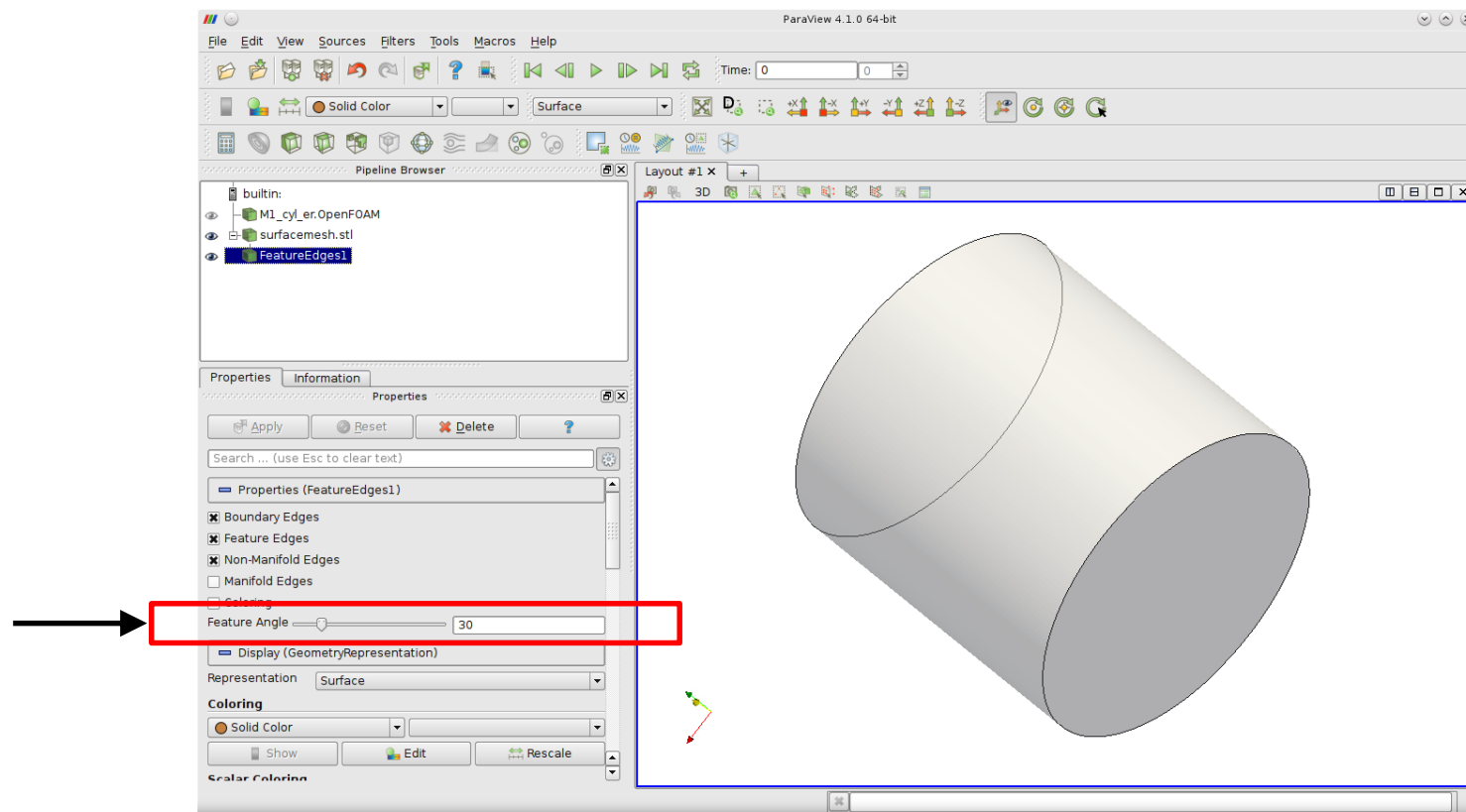
Mark edges whose adjacent surface normals
are at an angle less than includedAngle

0: selects no edges
180: selects all edge

snappyHexMesh guided tutorials

3D Cylinder with edge refinement.

- If you want to have a visual representation of the feature edges, you can use paraview/paraFoam.
- Just look for the filter `Feature Edges`.
- Have in mind that the angle you need to define in paraview/paraFoam is the complement of the angle you define in the dictionary `surfaceFeatureExtractDict`



snappyHexMesh guided tutorials

3D Cylinder with edge refinement.

- In this case we are going to generate a body fitted mesh with edge refinement. This is an external mesh.
- These are the dictionaries and files that will be used.
 - `system/snappyHexMeshDict`
 - `system/surfaceFeatureExtractDict`
 - `system/meshQualityDict`
 - `system/blockMeshDict`
 - `constant/triSurface/surfacemesh.stl`
 - `constant/triSurface/surfacemesh.eMesh`
- The file `surfacemesh.eMesh` is generated after using the utility `surfaceFeatureExtract`, which reads the dictionary `surfaceFeatureExtractDict`.
- The utility `surfaceFeatureExtract`, will save a set of *.obj files with the captured edges. These files are located in the directory `constant/extendedFeatureEdgeMesh`. You can use paraview to visualize the *.obj files.

snappyHexMesh guided tutorials

3D Cylinder with edge refinement.

- Let us generate the mesh, in the terminal window type:

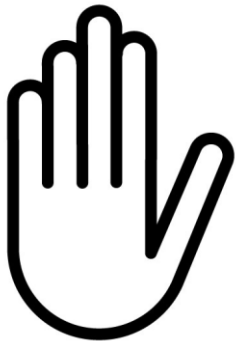
1. `$> foamCleanTutorials`
2. `$> surfaceFeatureExtract`
3. `$> blockMesh`
4. `$> snappyHexMesh -overwrite`
5. `$> checkMesh -latestTime`
6. `$> paraFoam`

- In step 2 we extract the sharp angles from the geometry.
- In step 3 we generate the background mesh.
- In step 4 we generate the body fitted mesh. Have in mind that as we use the option `-overwrite`, we are not saving the intermediate steps.
- In step 5 we check the mesh quality.

snappyHexMesh guided tutorials

- Meshing with snappyHexMesh.
- Meshing case 2. 3D Cylinder (external mesh), with feature edge refinement and boundary layer.

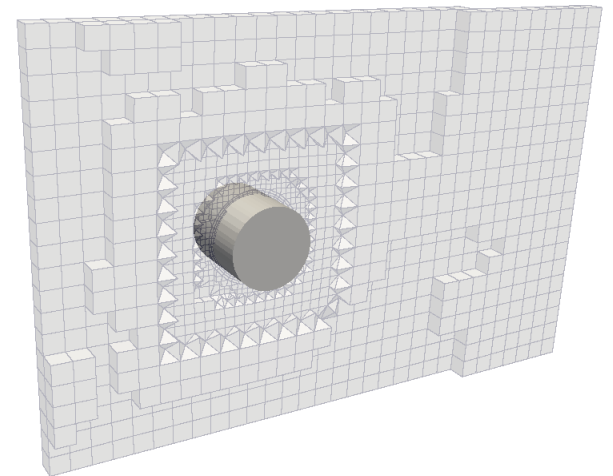
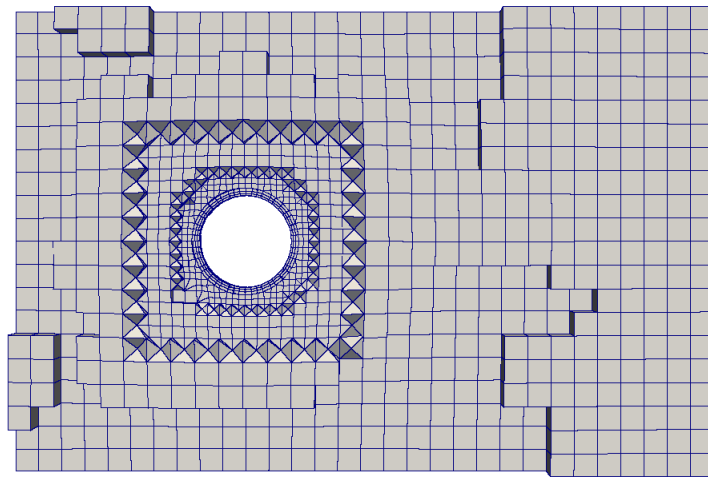
M1cy1/C2



- From this point on, please follow me.
- We are all going to work at the same pace.
- Remember, \$PTOFC is pointing to the path where you unpacked the tutorials.

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer.



Your final mesh should look like this one

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer.

- How do we enable boundary layer?
- In the file *snappyHexMeshDict*, look for the following entry:

```
castellatedMesh true;           //or false
snap true;                     //or false
addLayers true;                //or false
```

```
...
...
...
```

Set this parameter to true if you want to enable boundary layer meshing



snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer.

- How do we enable boundary layer?
- In the file *snappyHexMeshDict*, look for the section **addLayersControls**:

```
addLayersControls
```

```
{
```

```
    //Global parameters
```

```
    relativeSizes true;
```

```
    expansionRatio 1.2;
```

```
    finalLayerThickness 0.5;
```

```
    minThickness 0.1;
```

```
    layers
```

```
    {
```

```
        banana_stlSurface
```

```
        {
```

```
            nSurfaceLayers 3;
```

```
        }
```

```
    }
```

```
    // Advanced settings
```

```
    ...
```

```
    ...
```

```
    ...
```

```
}
```

← Name of the surface or user-defined patch where you want to add the boundary layer mesh.

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer.

- How do we control boundary layer collapsing?
- In the file *snappyHexMeshDict*, look for the section **addLayersControls**:

```
addLayersControls
```

```
{
```

```
...
```

```
...
```

```
...
```

```
// Advanced settings
```

```
nGrow 0;
```

```
featureAngle 130;
```

← Increase this value to avoid BL collapsing

```
...
```

```
...
```

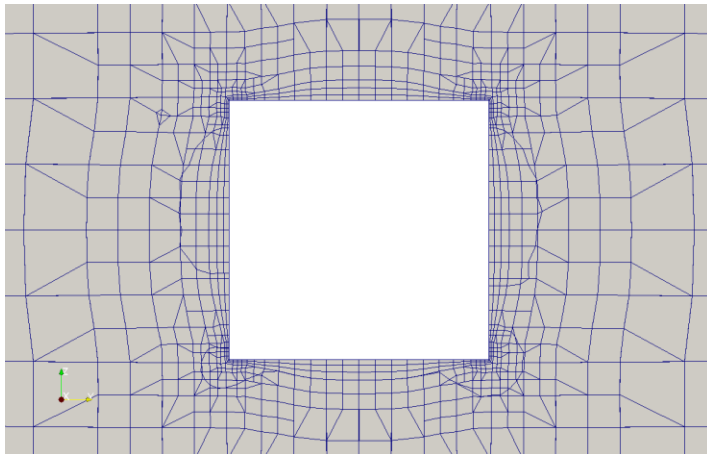
```
...
```

```
}
```

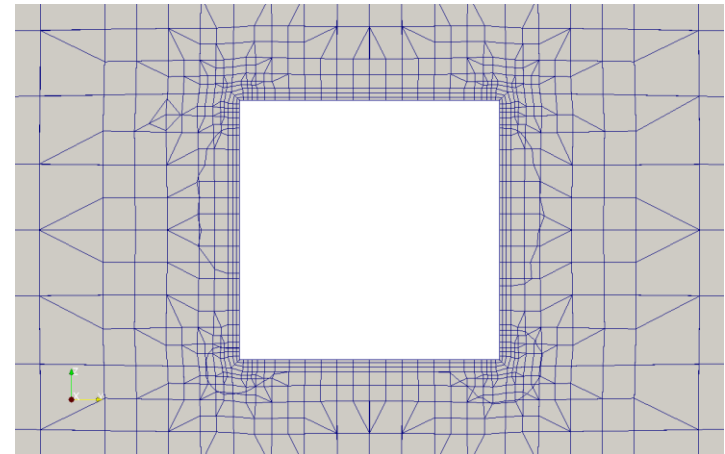
snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer.

Effect of different parameters on the boundary layer meshing



relativeSizes true
expansionRatio 1.2
finalLayerThickness 0.5
minThickness 0.1
featureAngle 130
nSurfaceLayers 3
Surface based refinement level (2 4)

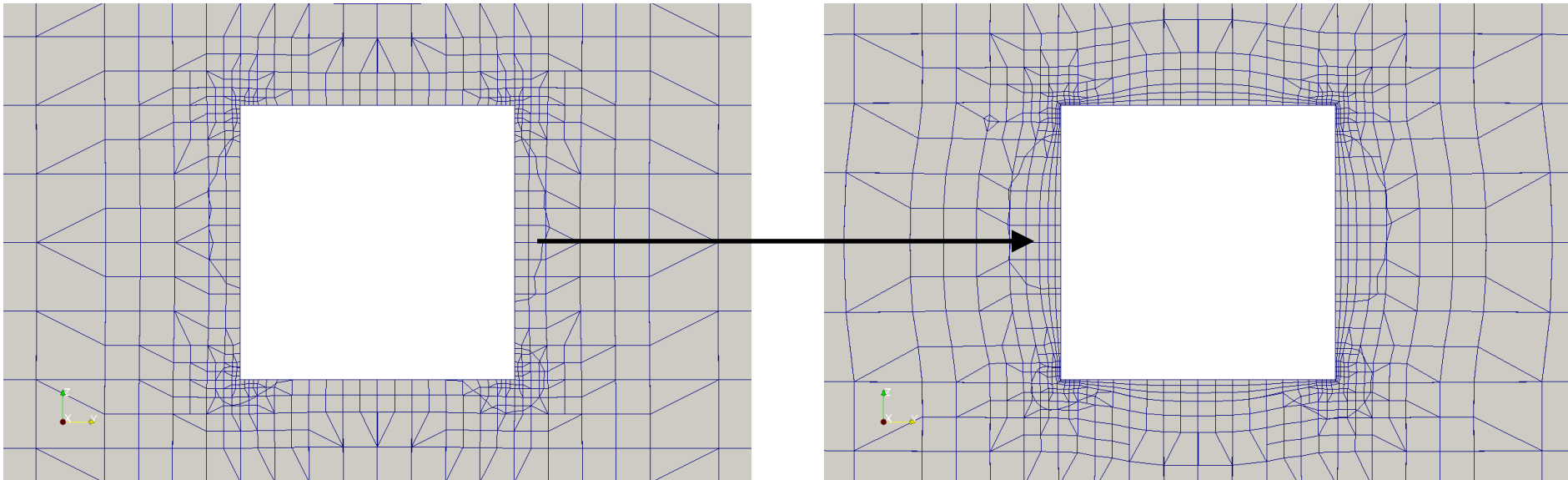


relativeSizes false
expansionRatio 1.2
firstLayerThickness 0.025
minThickness 0.01
featureAngle 130
nSurfaceLayers 3
Surface based refinement level (2 4)

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer.

Effect of different parameters on the boundary layer meshing

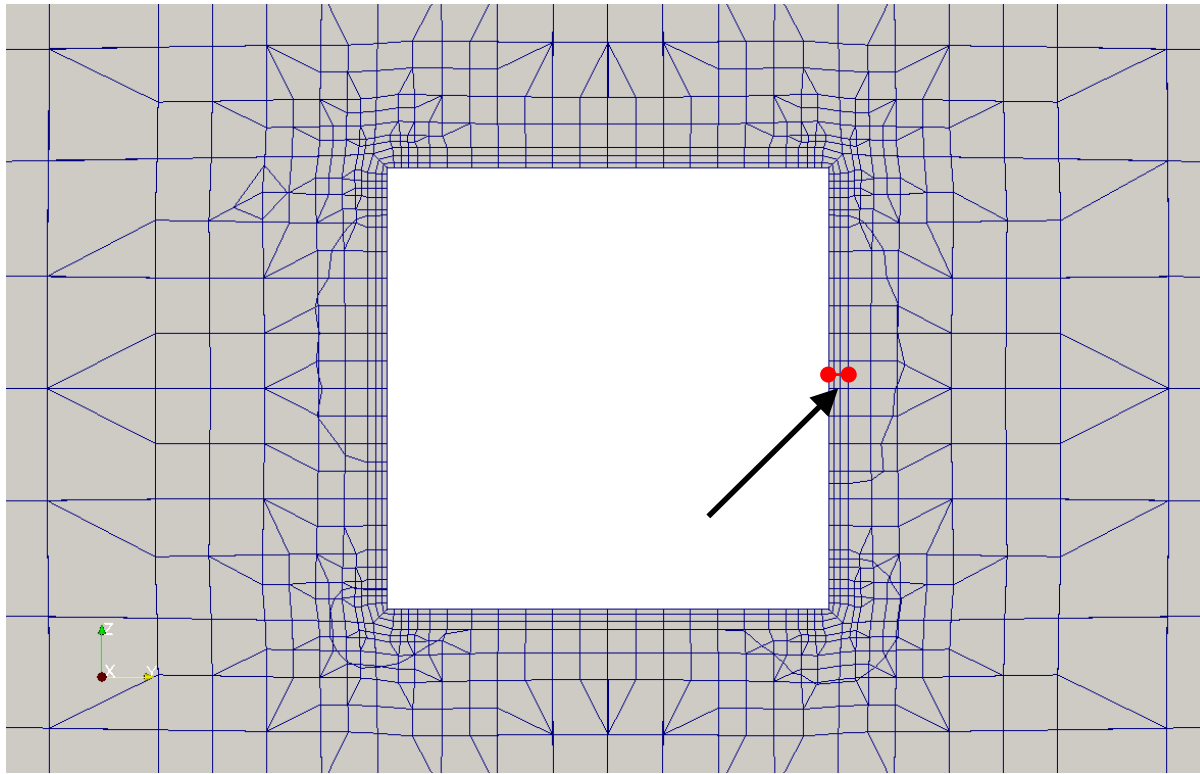


- When the option **relativeSizes** is true, the boundary layer meshing is done relative to the size of the cells next to the surface.
- This option requires less user intervention but can not guarantee a uniform boundary layer.
- Also, it is quite difficult to set a desired thickness of the first layer.

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer.

Effect of different parameters on the boundary layer meshing

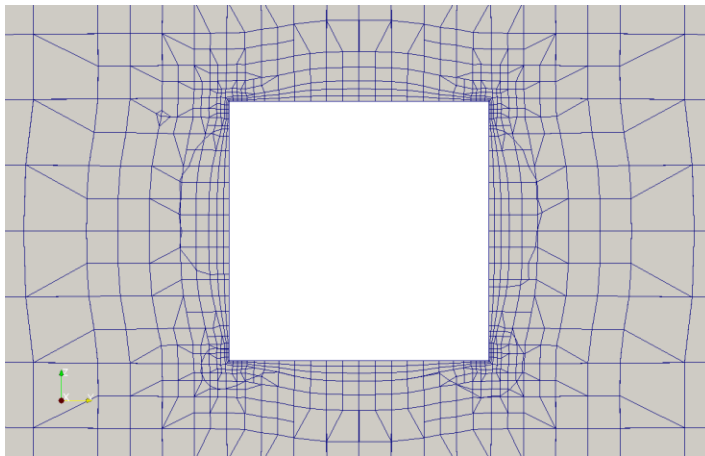


- When the option **relativeSizes** is false, we give the actual thickness of the layers.
- This option requires a lot user intervention but it guarantees a uniform boundary layer and the desired layer thickness.

snappyHexMesh guided tutorials

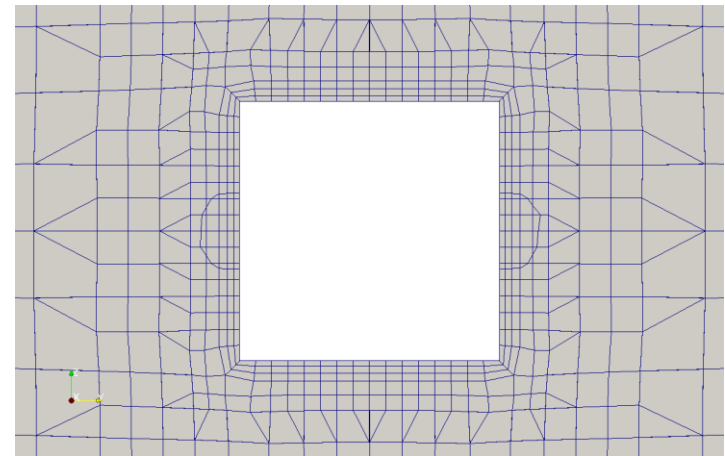
3D Cylinder with edge refinement and boundary layer.

Effect of different parameters on the boundary layer meshing



relativeSizes true
expansionRatio 1.2
finalLayerThickness 0.5
minThickness 0.1
featureAngle 130
nSurfaceLayers 3

Surface based refinement level (2 4)



relativeSizes true
expansionRatio 1.2
finalLayerThickness 0.5
minThickness 0.1
featureAngle 130
nSurfaceLayers 3

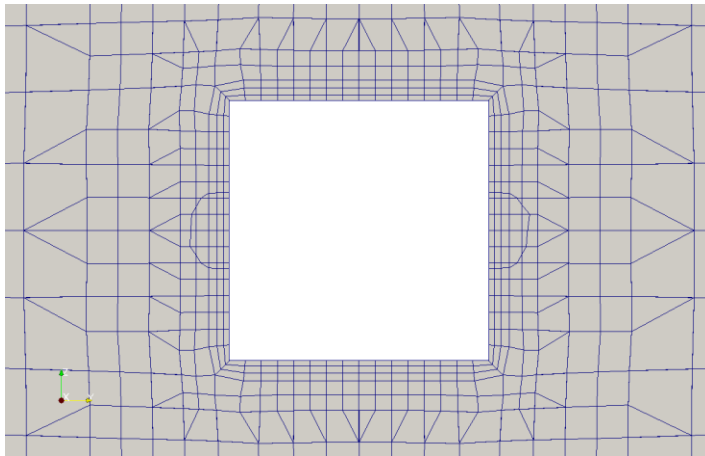
Surface based refinement level (2 2)

- When the option **relativeSizes** is true and in order to have a uniform boundary layer, we need to have a uniform surface refinement.
- Nevertheless, we still do not have control on the desired thickness of the first layer.

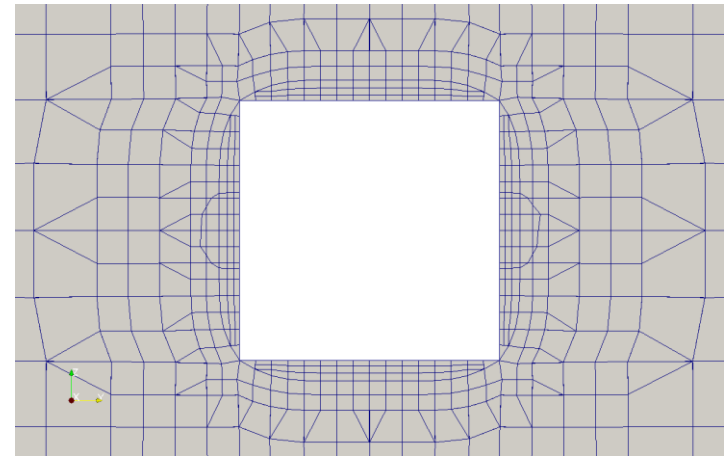
snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer.

Effect of different parameters on the boundary layer meshing



relativeSizes true
expansionRatio 1.2
finalLayerThickness 0.5
minThickness 0.1
featureAngle 130
nSurfaceLayers 3
Surface based refinement level (2 2)



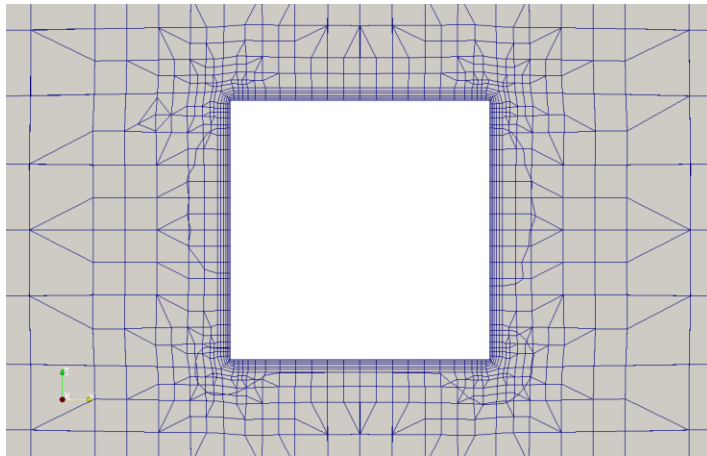
relativeSizes true
expansionRatio 1.2
finalLayerThickness 0.5
minThickness 0.1
featureAngle 30
nSurfaceLayers 3
Surface based refinement level (2 2)

- To avoid boundary layer collapsing close to the corners, we can increase the value of the boundary layer parameter **featureAngle**.

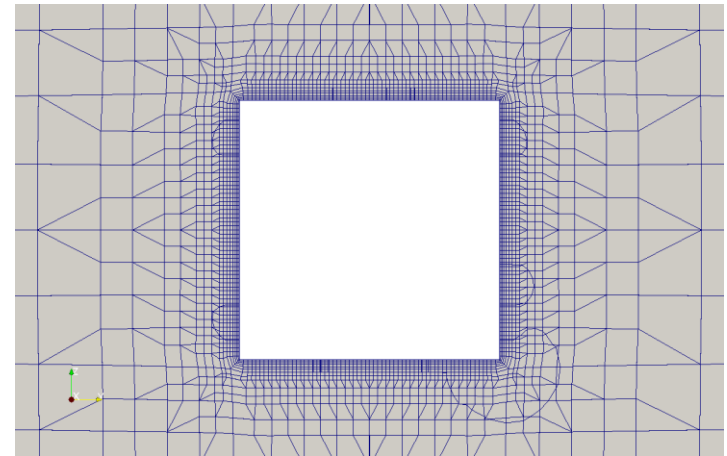
snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer.

Effect of different parameters on the boundary layer meshing



relativeSizes false
nSurfaceLayers 6



relativeSizes false
nSurfaceLayers 6

Refinement region at the stl surface:
mode distance;
levels ((0.05 4))

- The disadvantage of setting **relativeSizes** to false, is that it is difficult to control the expansion ratio from the boundary layer meshing to the far mesh.
- To control this transition, we can add a refinement region at the surface with distance mode.

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer.

- To generate the mesh, in the terminal window type:

1. `$> foamCleanTutorials`
2. `$> surfaceFeatureExtract`
3. `$> blockMesh`
4. `$> snappyHexMesh -overwrite`
5. `$> checkMesh -latestTime`
6. `$> paraFoam`

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer.

- At the end of the meshing process you will get the following information regarding the boundary layer meshing:

patch	faces	layers	overall [m]	thickness [%]
-----	-----	-----	---	---
banana_stlSurface	4696	3	0.0569	95.9
Layer mesh : cells:48577 faces:157942 points:61552				

- This is a general summary of the boundary layer meshing.
- Pay particular attention to the overall and thickness information.
- Overall is roughly speaking the thickness of the whole boundary layer.
- Thickness is the percentage of the patch that has been covered with the boundary layer mesh. A thickness of 100% means that the whole patch has been covered (a perfect BL mesh).

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer.

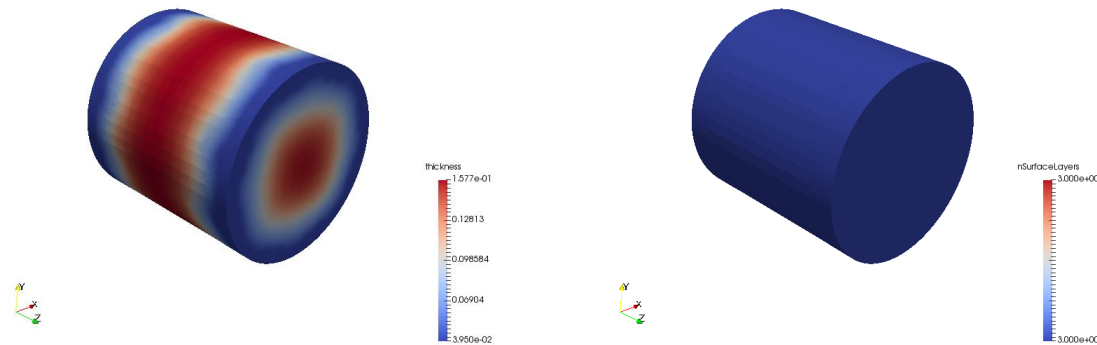
- If you want to visualize the boundary layer thickness, you can enable **writeFlags** in the *snappyHexMeshDict* dictionary,

```
...  
...  
...  
  
writeFlags  
(  
    scalarLevels; // write volScalarField with cellLevel for postprocessing  
    layerSets;    // write cellSets, faceSets of faces in layer  
    layerFields;  // write volScalarField for layer coverage  
);  
  
...  
...  
...
```

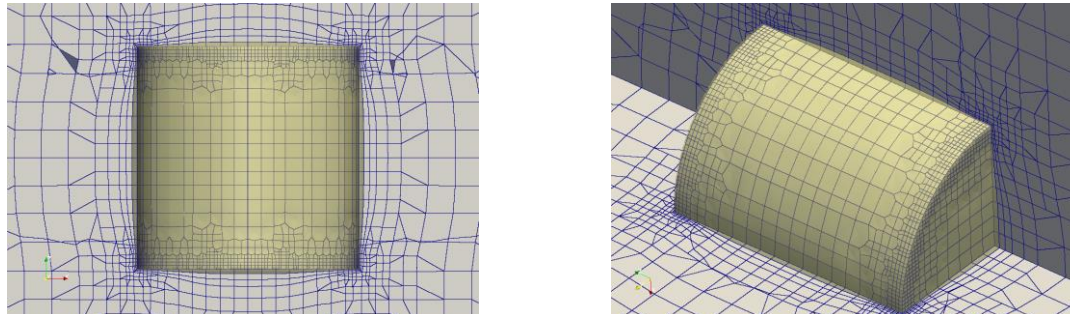
snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer.

- Then you can use paraview/paraFoam to visualize the boundary layer coverage.



Boundary layer thickness and number of layers



The yellow surface represent the BL coverage

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer.

- After creating the mesh and if you do not like the inflation layer or you want to try different layer parameters, you do not need to start the meshing process from scratch.
- To restart the meshing process from a saved state you need to save the intermediate steps (castellation and snapping), and then create the inflation layers starting from the snapped mesh.
- That is, do not use the option `snappyHeshMesh -overwrite`.
- Also, in the dictionary `controlDict` remember to set the entry `startFrom` to `latestTime` or the time directory where the snapped mesh is saved (in this case 2).
- Before restarting the meshing, you will need to turn off the castellation and snapping options and turn on the boundary layer options in the `snappyHexMeshDict` dictionary.

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer.

- Remember, before restarting the meshing you will need to modify the *snappyHexMeshDict* dictionary as follows:

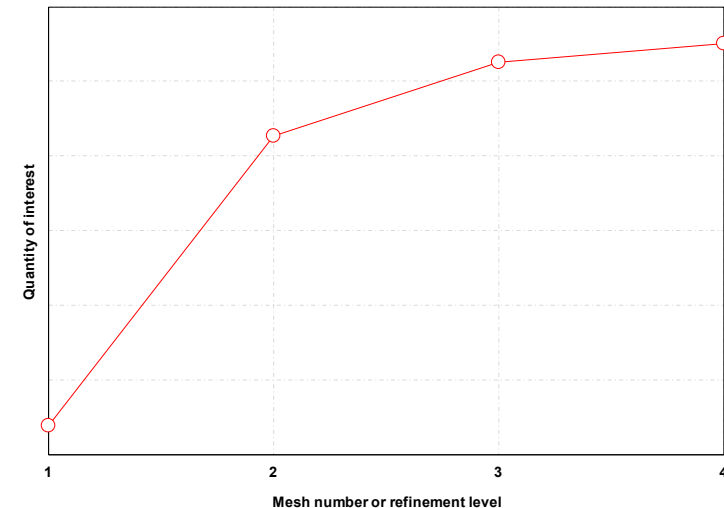
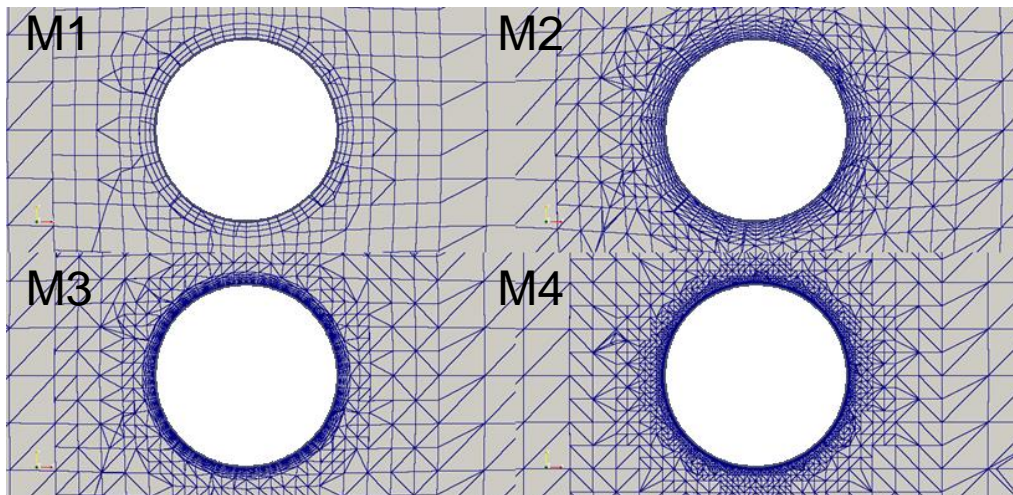
```
castellatedMesh    false;  
snap               false;  
addLayers          true;
```

- At this point, you can restart the meshing process by typing in the terminal,
 - \$> snappyHexMesh
- By the way, you can restart the boundary layer mesh from a previous mesh with a boundary layer.
- So in theory, you can add one layer at a time, this will give you more control but it will require more manual work and some scripting.

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer.

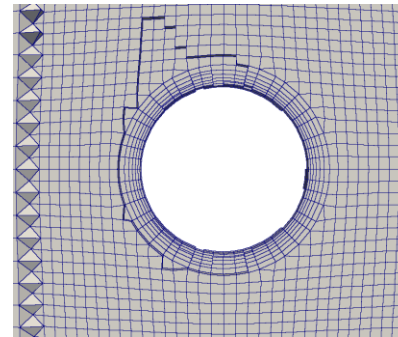
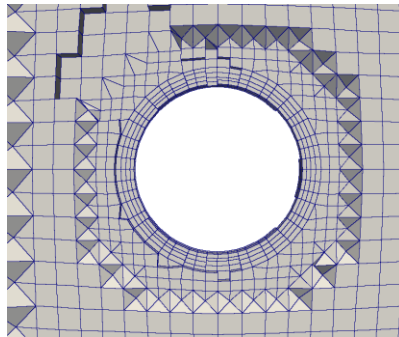
- How do I know if my mesh is OK?
 - If your goal is to predict the forces at the walls, you should have a surface mesh and boundary layer mesh fine enough to resolve well the forces.
 - Also, if there is transition to turbulence you should resolve very well the boundary layer.
 - To have an idea if you are resolving well the forces, you can do a grid refinement study. Have in mind that these studies can be really expensive.



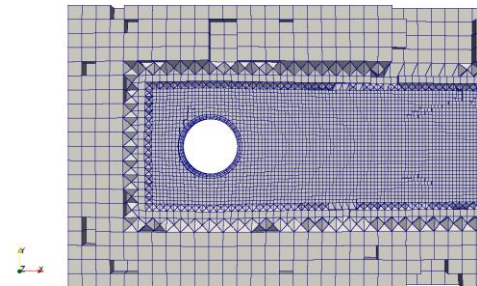
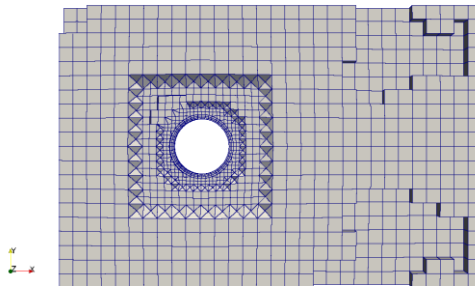
snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer.

- How do I know if my mesh is OK?
 - To avoid smearing gradients close to the walls, remember to control the transition from the boundary layer mesh to the far field mesh.



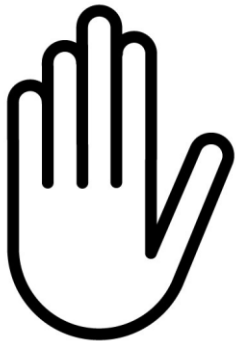
- And also you will need to have a mesh fine enough to resolve the flow features far from the body.



snappyHexMesh guided tutorials

- Meshing with snappyHexMesh.
- Meshing case 3. 3D Cylinder (external mesh), with feature edge refinement and boundary layer, using a STL file with multiple surfaces.

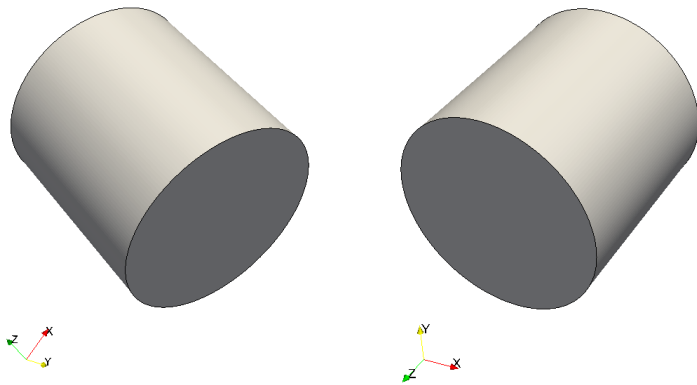
M1cy1/C3



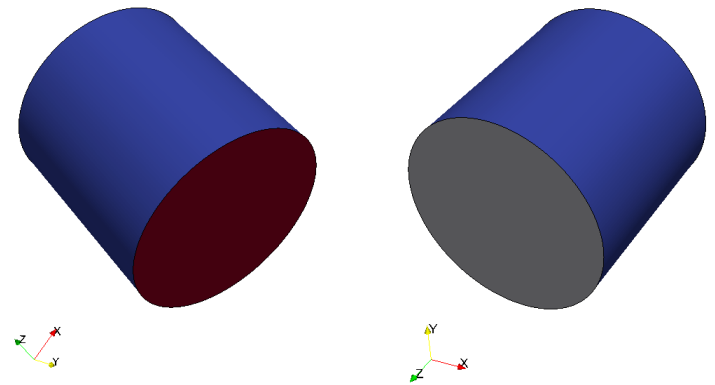
- From this point on, please follow me.
- We are all going to work at the same pace.
- Remember, \$PTOFC is pointing to the path where you unpacked the tutorials.

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer, using a STL file with multiple surfaces.



STL visualization with a single surface using paraview (the single surface is represented with a single color)

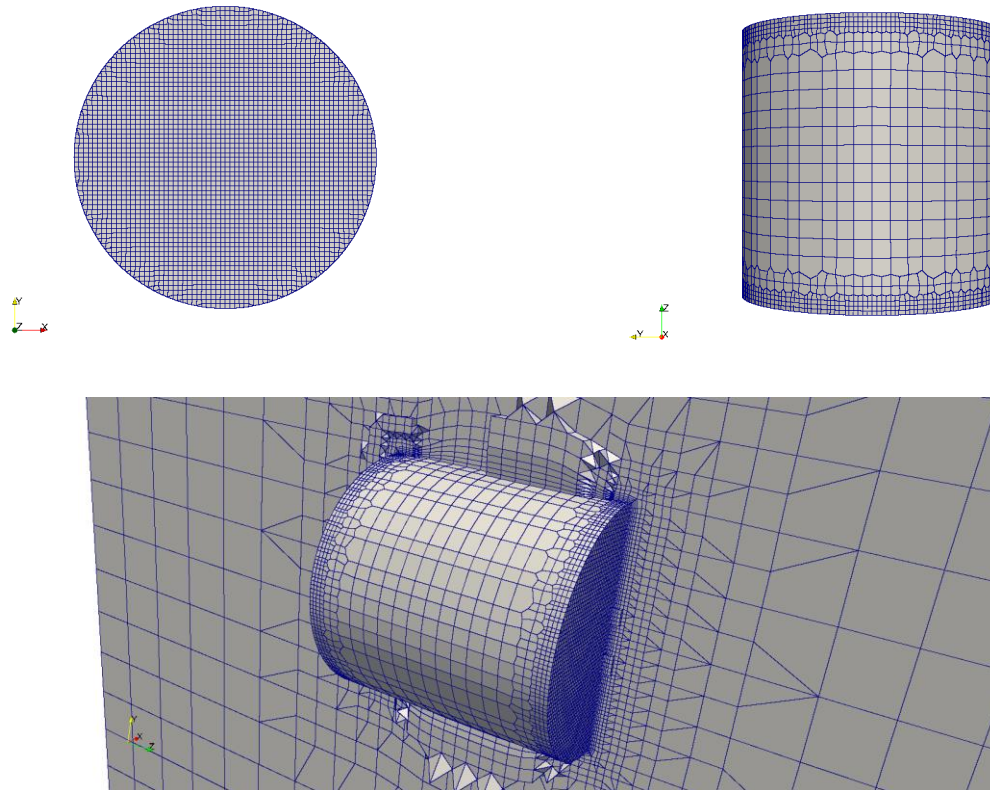


STL visualization with multiple surfaces using paraview (each color corresponds to a different surface)

- When you use a STL with multiple surfaces, you have more control over the meshing process.
- By default, STL files are made up of one single surface.
- If you want to create the multiple surfaces you will need to do it in the solid modeler.
- Alternatively, you can split the STL manually or using the utility `surfaceAutoPatch`.
- Loading multiple STLs is equivalent to using a STL with multiple surfaces.

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer, using a STL file with multiple surfaces.



- When you use a STL with multiple surfaces, you have more control over the meshing process.
- In this case, we were able to use different refinement parameters in the lateral and central surface patches of the cylinder.

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer, using a STL file with multiple surfaces.

- How do we assign different names to different surface patches?
- In the file *snappyHexMeshDict*, look for the following entry:

```
geometry
{
    surfacemesh.stl
    {
        type triSurfaceMesh;
        name stlSurface;

        regions
        {
            patch0
            {
                name surface0;
            }
            patch1
            {
                name surface1;
            }
            patch2
            {
                name surface2;
            }
        }
    }
    ...
    ...
    ...
}
```

Named region in the STL file

User-defined patch name
This is the name you need to use when setting the boundary layer meshing

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer, using a STL file with multiple surfaces.

- How do we refine user defined surface patches?
- In the file *snappyHexMeshDict*, look for the following entry:

```
castellatedMeshControls
{
    ...
    ...
    ...
    refinementSurfaces
    {
        level (2 2); ← Global refinement level
        regions
        {
            patch0 ← Local surface patch
            {
                level (2 2); ← Local refinement level
                patchInfo
                {
                    type wall; ← Type of the patch.
                                This information is optional
                }
            }
            ...
            ...
            ...
        }
    }
    ...
    ...
    ...
}
```

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer, using a STL file with multiple surfaces.

- How do we control curvature refinement on surface patches?
- In the file *snappyHexMeshDict*, look for the following entry:

```
castellatedMeshControls
{
    ...
    ...
    ...
    refinementSurfaces
    {
        level (2 2); ← Global refinement level
        regions
        {
            patch0 ← Local surface patch
            {
                level (2 4); ← Local curvature refinement (in red)
                patchInfo
                {
                    type wall;
                }
            }
            ...
            ...
            ...
        }
    }
    ...
    ...
    ...
}
```

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer, using a STL file with multiple surfaces.

- How do we control curvature refinement on surface patches?
- In the file *snappyHexMeshDict*, look for the following entry:

```
castellatedMeshControls
```

```
{
```

```
...
```

```
...
```

```
...
```

```
//Local curvature and  
//feature angle refinement  
resolveFeatureAngle 60;
```

The default value is 30.
Using a higher value will capture
less features.

```
...
```

```
...
```

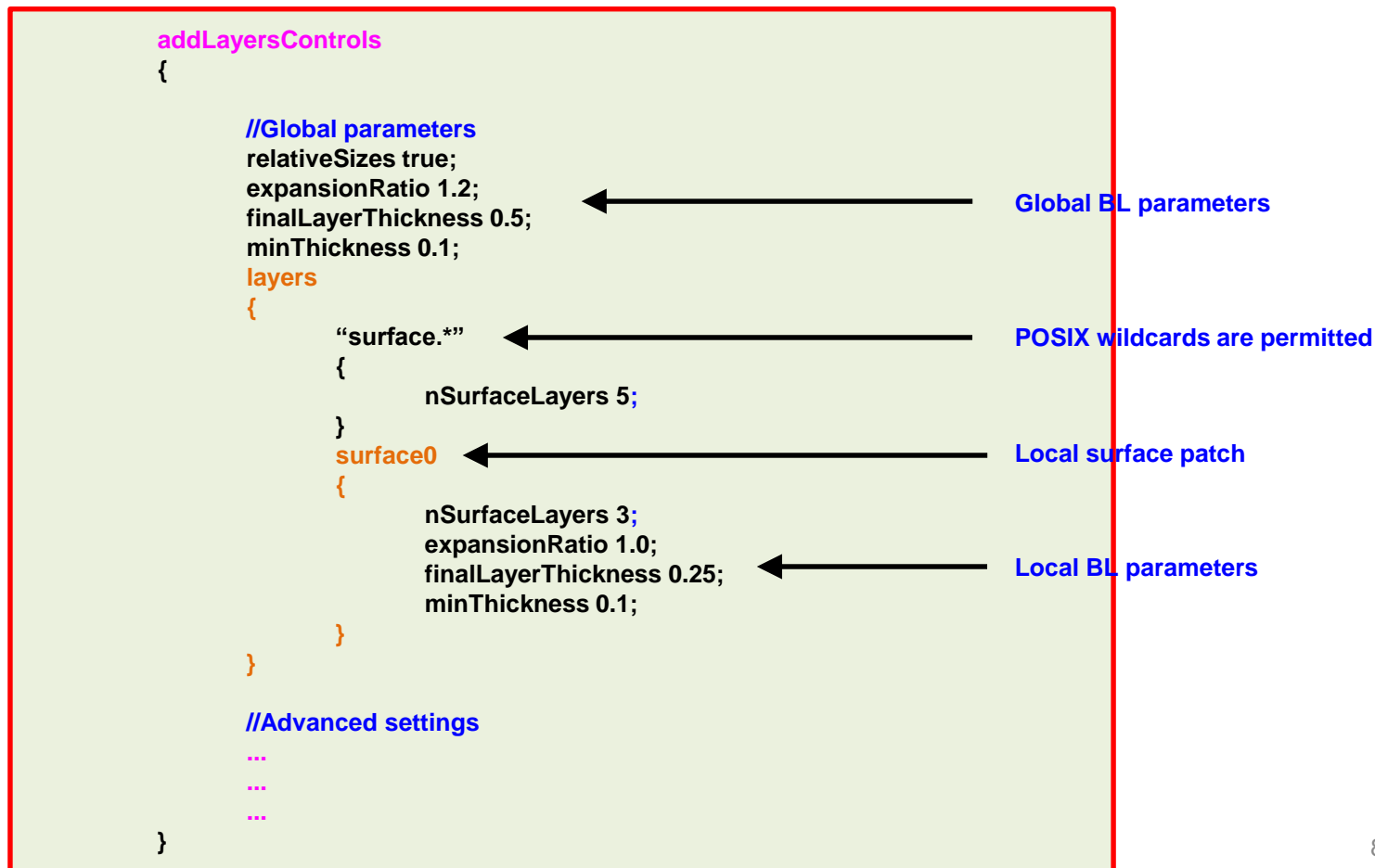
```
...
```

```
}
```

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer, using a STL file with multiple surfaces.

- How do we control boundary layer meshing on the surface patches?
- In the file *snappyHexMeshDict*, look for the following entry:



snappyHexMesh guided tutorials

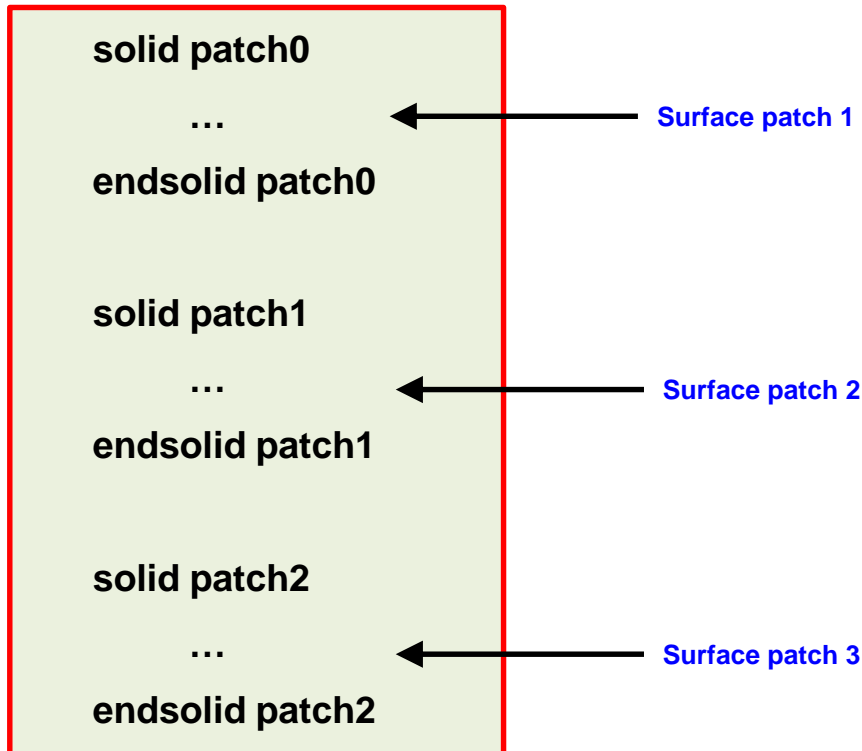
3D Cylinder with edge refinement and boundary layer, using a STL file with multiple surfaces.

- Let us first create the STL file with multiple surfaces.
- In the directory **geo**, you will find the original STL file.
- In the terminal type:
 1. `$> cd geo`
 2. `$> surfaceAutoPatch geo.stl output.stl 130`
 3. `$> cp output.stl ../constant/triSurface/surfacemesh.stl`
 4. `$> cd ..`
 5. `$> paraview`
- The utility `surfaceAutoPatch` will read the original STL file (`geo.stl`), and it will find the patches using an angle criterion of 130 (similar to the angle criterion used with the utility `surfaceFeatureExtract`). It writes the new STL geometry in the file `output.stl`.
- By the way, it is better to create the STL file with multiple surfaces directly in the solid modeler.
- FYI, there is an equivalent utility for meshes, `autoPatch`. So if you forgot to define the patches, this utility will automatically find the patches according to an angle criterion.

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer, using a STL file with multiple surfaces.

- If you open the file *output.stl*, you will notice that there are three surfaces defined in the STL file. The different surfaces are defined in by the following sections:



- The name of the solid sections are automatically given by the utility `surfaceAutoPatch`.
- The convention is as follows: patch0, patch1, patch2, ... patchN.
- If you do not like the names, you can change them directly in the STL file.

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer, using a STL file with multiple surfaces.

- The new STL file is already in the `constant/triSurface` directory.
- To generate the mesh, in the terminal window type:

1. `$> foamCleanTutorials`
2. `$> surfaceFeatureExtract`
3. `$> blockMesh`
4. `$> snappyHexMesh -overwrite`
5. `$> checkMesh -latestTime`

- To visualize the mesh, in the terminal window type:

6. `$> paraFoam`

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer, using a STL file with multiple surfaces.

- This case is ready to run using the solver `simpleFoam`. But before running, you will need to set the boundary and initial conditions.
- You will need to manually modify the file `constant/polyMesh/boundary`
- Remember:
 - **Base type** boundary conditions are defined in the file `boundary` located in the directory `constant/polyMesh`.
 - **Primitive** or **numerical type** boundary conditions are defined in the field variables files located in the directory `0` or the time directory from which you want to start the simulation (e.g. U , p).
 - The name of the base type boundary conditions and numerical type boundary conditions needs to be the same.
 - Also, the base type boundary condition needs to be compatible with the numerical type boundary condition.

snappyHexMesh guided tutorials

3D Cylinder with edge refinement and boundary layer, using a STL file with multiple surfaces.

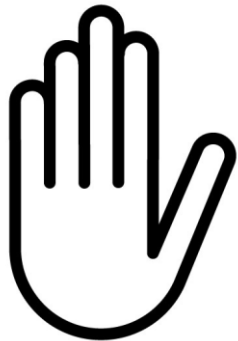
- This case is ready to run with `simpleFoam`.
- If you are in a hurry and you do not want to create/edit the files `0/U`, `0/p`, `constant/polyMesh/boundary`, follow these steps to run the case:

1. `$> cp 0_org/* 0`
2. `$> cp system/boundary_org constant/polyMesh/boundary`
3. `$> renumberMesh -overwrite`
4. `$> simpleFoam > log | tail -f log`

snappyHexMesh guided tutorials

- Meshing with snappyHexMesh.
- Meshing case 4. 2D Cylinder (external mesh).

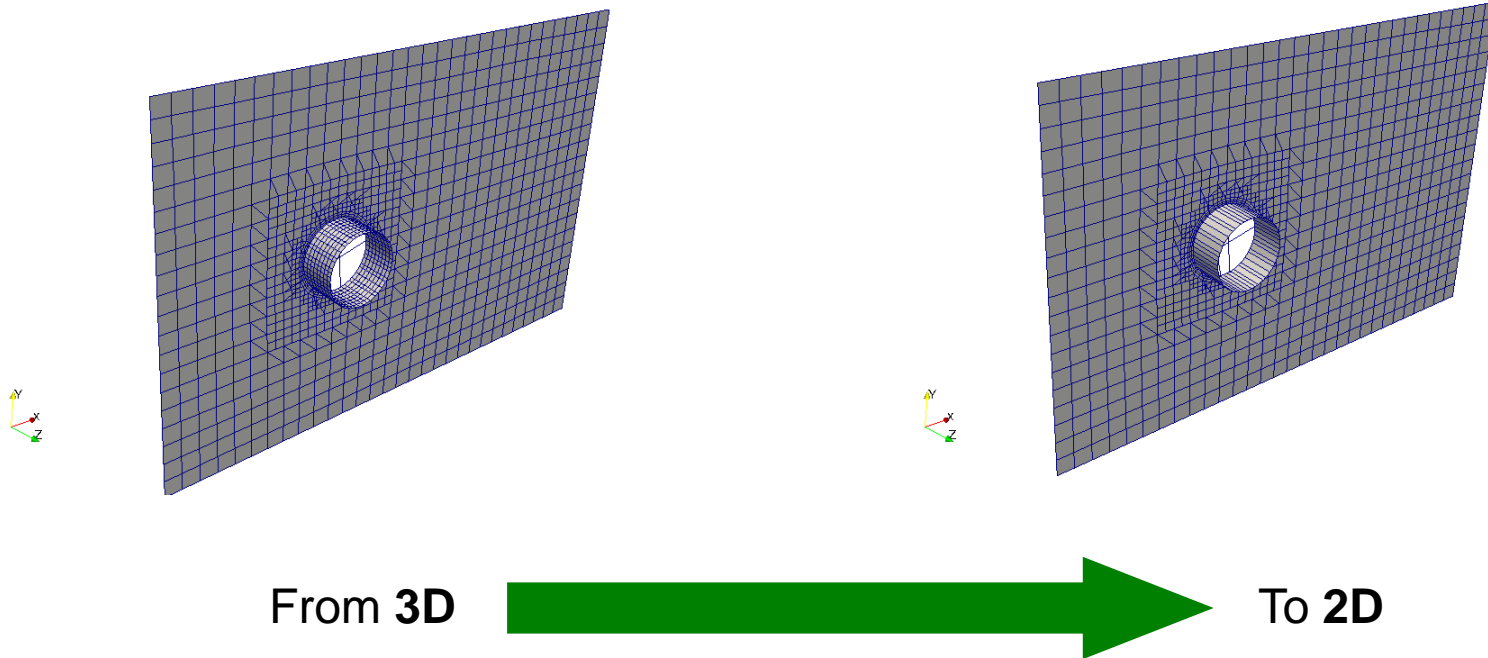
M1cy1/C4



- From this point on, please follow me.
- We are all going to work at the same pace.
- Remember, \$PTOFC is pointing to the path where you unpacked the tutorials.

snappyHexMesh guided tutorials

2D Cylinder

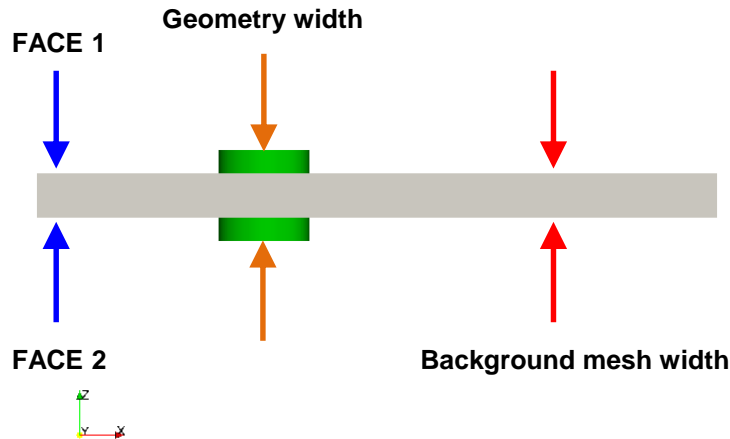


- To generate a 2D mesh using `snappyHexMesh`, we need to start from a 3D. After all, `snappyHexMesh` is a 3D mesher.
- To generate a 2D mesh (and after generating the 3D mesh), we use the utility `extrudeMesh`.
- The utility `extrudeMesh` works by projecting a face into a mirror face. Therefore, the faces need to be parallel.

snappyHexMesh guided tutorials

2D Cylinder

The utility `extrudeMesh` works by projecting FACE 1 into FACE 2. Therefore, the faces need to be parallel.



- At most, the input geometry and the background mesh need to have the same width.
- If the input geometry is larger than the background mesh, it will be automatically cut by the faces of the background mesh.
- In this case, the input geometry will be cut by the two lateral patches of the background mesh.
- If you want to take advantage of symmetry in 3D, you can cut the geometry in half using one of the faces of the background mesh.

snappyHexMesh guided tutorials

2D Cylinder

- How do we create the 2D mesh?
- After generating the 3D mesh, we use the utility `extrudeMesh`.
- This utility reads the `extrudeMeshDict`,

```
constructFrom patch;
```

```
sourceCase “.”
```

```
sourcePatches (minZ);
```



Name of source patch

```
exposedPatchName maxZ;
```



Name of the mirror patch

```
extrudeModel linearNormal
```

```
nLayers 1;
```



Number of layers to use in the linear extrusion.
As this is a 2D case we must use 1 layer

```
linearNormalCoeffs
```

```
{
```

```
    thickness 1;
```



Thickness of the extrusion.
It is highly recommended to use a value of 1

```
}
```

```
mergeFaces false;
```


snappyHexMesh guided tutorials

2D Cylinder

- To generate the mesh, in the terminal window type:

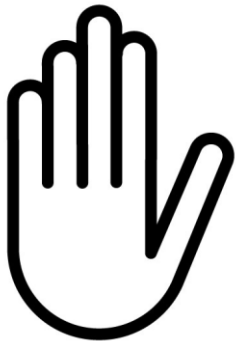
1. `$> foamCleanTutorials`
2. `$> blockMesh`
3. `$> snappyHexMesh -overwrite`
4. `$> extrudeMesh`
5. `$> checkMesh -latestTime`
6. `$> paraFoam`

- Remember, the utility `extrudeMesh` (step 4) reads the dictionary `extrudeMeshDict`, which is located in the directory **system**.

snappyHexMesh guided tutorials

- Meshing with snappyHexMesh.
- Meshing case 5. Ahmed body (external mesh)

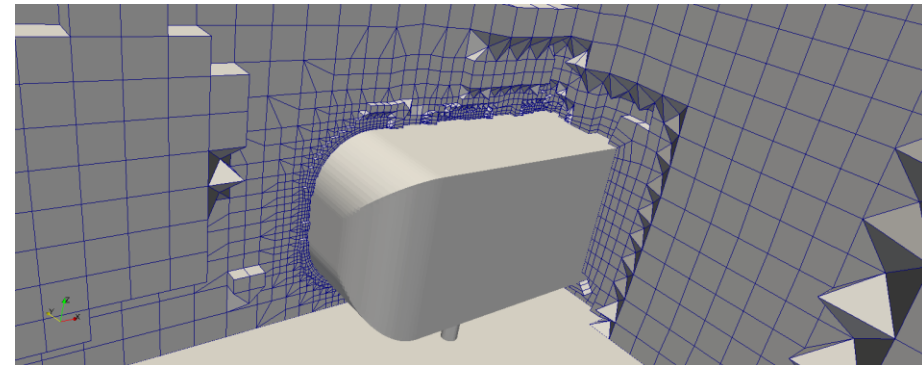
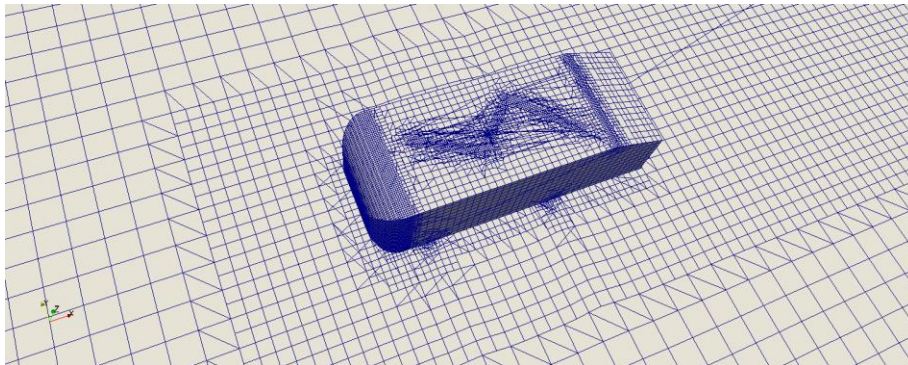
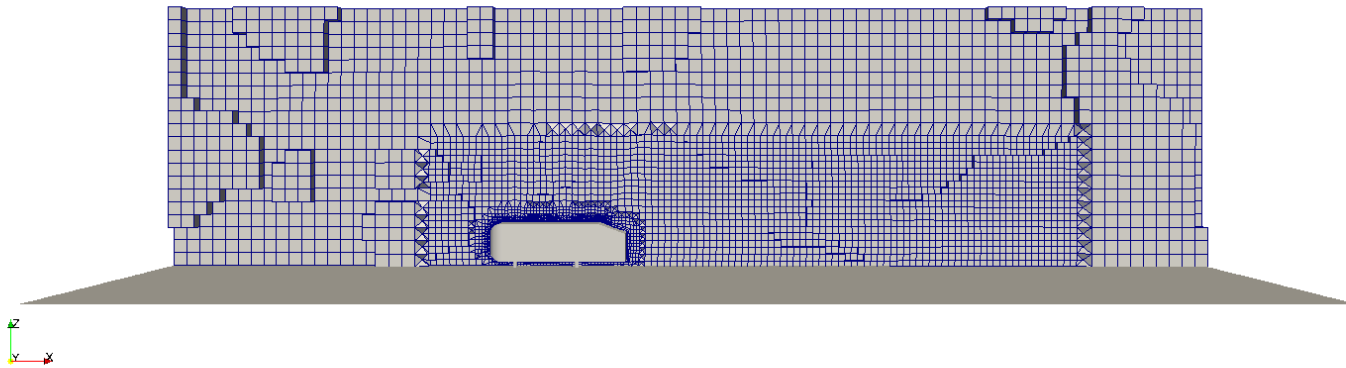
M3_ahmed



- From this point on, please follow me.
- We are all going to work at the same pace.
- Remember, \$PTOFC is pointing to the path where you unpacked the tutorials.

snappyHexMesh guided tutorials

Ahmed body



- At this point, we all have a clear idea of how `snappyHexMesh` works.
- If not, please raise your hand.
- So let us go free styling and let us play around with this case.

snappyHexMesh guided tutorials

Ahmed body

- In our YouTube channel you will find a playlist with many videos for this case. The playlist is titled: CFD workflow tutorial using open-source tools.
- You can find our YouTube channel in the following link:
<https://www.youtube.com/channel/UCNNBm3KxVS1rGeCVUU1p61g>
- In these videos, we show a few extra features and some tips and tricks to take the most out of snappyHexMesh.
- If you get lost, read the *README.FIRST* file that you will find in the working directory.
- The dictionaries *snappyHexMeshDict* and *blockMeshDict* used in this case are very clean and ready to use. So feel free to use them as your templates.
- Our best advice is not to get lost in all the options available in the dictionary *snappyHexMeshDict*. Most of the times the default options will work fine.
- That being said, you only need to read in the geometries, set the feature edges and surface refinement levels, choose in which surfaces you want to add the boundary layers, and choose how many layers you want to add.
- Final advices:
 - If you are working with a complicated geometry, add one layer at a time.
 - Use paraFoam/paraview to get visual references.
 - Always check the quality of your mesh.