OpenFOAM User Guide: 5.4 Mesh generation with snappyHexMesh

[Table of Contents][Index]

[prev] [next]

5.4 Mesh generation with the *snappyHexMesh* utility

This section describes the mesh generation utility, *snappyHexMesh*, supplied with OpenFOAM. The *snappyHexMesh* utility generates 3-dimensional meshes containing hexahedra (hex) and split-hexahedra (split-hex) automatically from triangulated surface geometries, or tri-surfaces, in Stereolithography (STL) or Wavefront Object (OBJ) format. The mesh approximately conforms to the surface by iteratively refining a starting mesh and morphing the resulting split-hex mesh to the surface. An optional phase will shrink back the resulting mesh and insert cell layers. The specification of mesh refinement level is very flexible and the surface handling is robust with a pre-specified final mesh quality. It runs in parallel with a load balancing step every iteration.

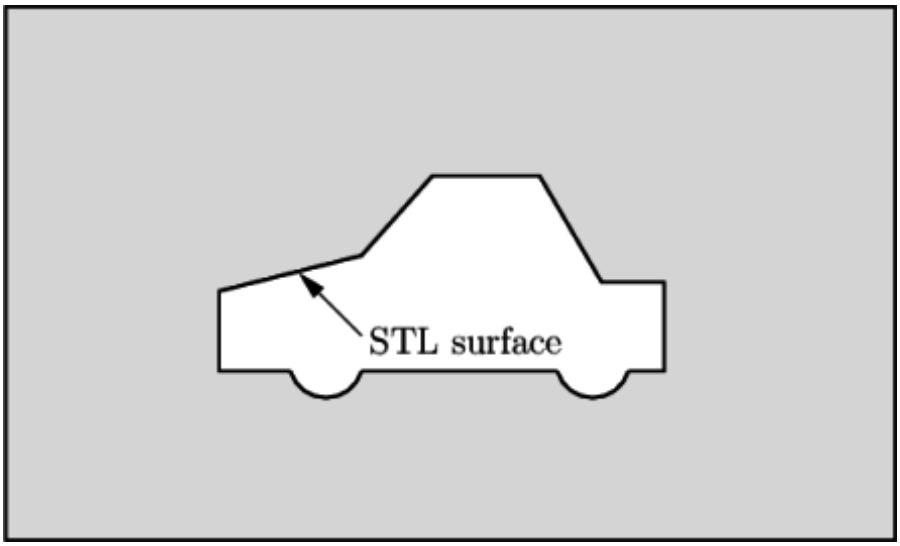

Figure 5.9: Schematic 2D meshing problem for `snappyHexMesh`

5.4.1 The mesh generation process of `snappyHexMesh`

The process of generating a mesh using `snappyHexMesh` will be described using the schematic in Figure 5.9. The objective is to mesh a rectangular shaped region (shaded grey in the figure) surrounding an object described by a tri-surface, `e.g.` typical for an external aerodynamics simulation. Note that the schematic is 2-dimensional to make it easier to understand, even though the `snappyHexMesh` is a 3D meshing tool.

In order to run `snappyHexMesh`, the user requires the following:

- one or more tri-surface files located in a `constant/triSurface` sub-directory of the case directory;
- a background hex mesh which defines the extent of the computational domain and a base level mesh density; typically generated using `blockMesh`, discussed in section 5.4.2.
- a `snappyHexMeshDict` dictionary, with appropriate entries, located in the `system` sub-directory of the case.

The `snappyHexMeshDict` dictionary includes: switches at the top level that control the various stages of the meshing process; and, individual sub-directories for each process. The entries are listed in Table 5.6.

Keyword	Description	Example
`castellatedMesh`	Create the castellated mesh?	`true`
`snap`	Do the surface snapping stage?	`true`

doLayers	Add surface layers?	true
mergeTolerance	Merge tolerance as fraction of bounding box of initial mesh	1e-06
debug	Controls writing of intermediate meshes and screen printing	
	— Write final mesh only	0
	— Write intermediate meshes	1
	— Write *volScalarField* with cellLevel for post-processing	2
	— Write current intersections as *.obj* files	4
geometry	Sub-dictionary of all surface geometry used	
castellatedMeshControls	Sub-dictionary of controls for castellated mesh	
snapControls	Sub-dictionary of controls for surface snapping	
addLayersControls	Sub-dictionary of controls for layer addition	
meshQualityControls	Sub-dictionary of controls for mesh quality	

Table 5.6: Keywords at the top level of *snappyHexMeshDict*.

All the geometry used by *snappyHexMesh* is specified in a *geometry* sub-dictionary in the *snappyHexMeshDict* dictionary. The geometry can be specified through a tri-surface or bounding geometry entities in OpenFOAM. An example is given below:

```
  geometry
  {
      sphere.stl // STL filename
      {
          type triSurfaceMesh;
          regions
          {
              secondSolid             // Named region in the STL file
              {
                  name mySecondPatch; // User-defined patch name
              }                       // otherwise given sphere.stl_s
econdSolid
          }
```

```
        }

        box1x1x1  // User defined region name
        {
            type   searchableBox;       // region defined by bounding b
ox
            min    (1.5 1 -0.5);
            max    (3.5 2 0.5);
        }

        sphere2  // User defined region name
        {
            type   searchableSphere;    // region defined by bounding s
phere
            centre (1.5 1.5 1.5);
            radius 1.03;
        }
    };
```

5.4.2 Creating the background hex mesh

Before *snappyHexMesh* is executed the user must create a background mesh of hexahedral cells that fills the entire region within by the external boundary as shown in Figure 5.10.

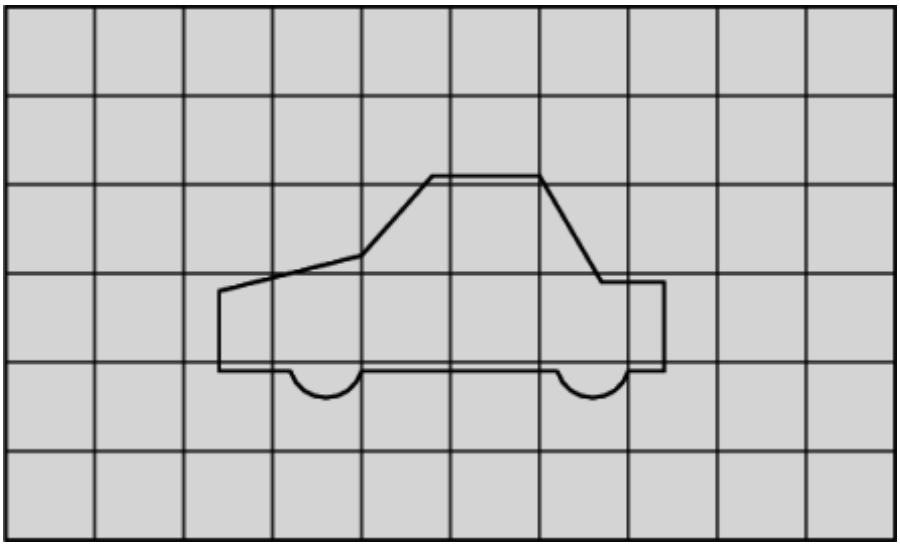

Figure 5.10: Initial mesh generation in *snappyHexMesh* meshing process

This can be done simply using *blockMesh*. 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 surfaces at

which the subsequent snapping procedure is applied, otherwise the convergence of the snapping procedure is slow, possibly to the point of failure;
- there must be at least one intersection of a cell edge with the tri-surface, *i.e.* a mesh of one cell will not work.

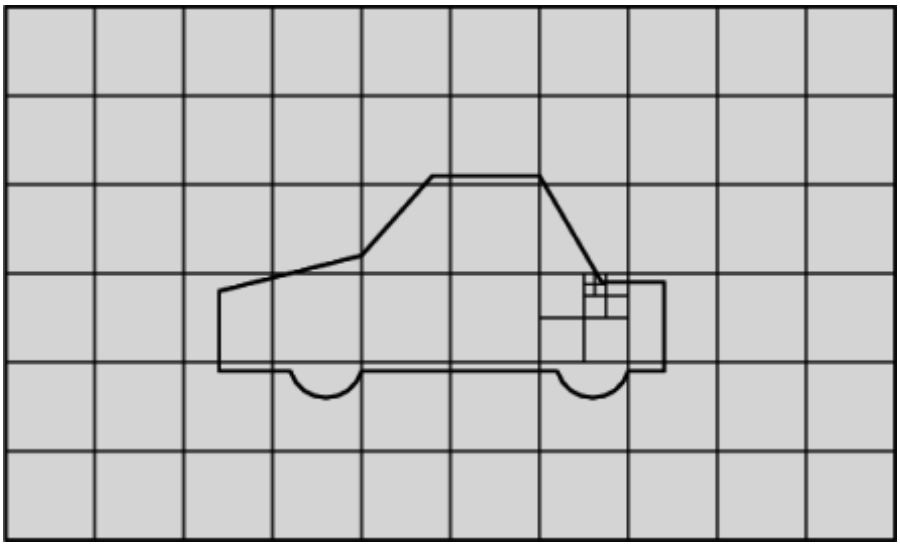

Figure 5.11: Cell splitting by feature edge in *snappyHexMesh* meshing process

5.4.3 Cell splitting at feature edges and surfaces

Cell splitting is performed according to the specification supplied by the user in the *castellatedMeshControls* sub-dictionary in the *snappyHexMeshDict*. The entries for *castellatedMeshControls* are presented in Table 5.7.

Keyword	Description	Example
locationInMesh	Location vector inside the region to be meshed	(5 0 0)
	N.B. vector must not coincide with a cell face either before or during refinement	
maxLocalCells	Max number of cells per processor during refinement	1e+06
maxGlobalCells	Overall cell limit during refinement (*i.e.* before removal)	2e+06
minRefinementCells	If ≥ number of cells to be refined, surface refinement stops	0

nCellsBetweenLevels	Number of buffer layers of cells between different levels of refinement	1
resolveFeatureAngle	Applies maximum level of refinement to cells that can see intersections whose angle exceeds this	30
features	List of features for refinement	
refinementSurfaces	Dictionary of surfaces for refinement	
refinementRegions	Dictionary of regions for refinement	

Table 5.7: Keywords in the *castellatedMeshControls* sub-dictionary of *snappyHexMeshDict*.

The splitting process begins with cells being selected according to specified edge features first within the domain as illustrated in Figure 5.11. The `features` list in the *castellatedMeshControls* sub-dictionary permits dictionary entries containing a name of an *edgeMesh* file and the `level` of refinement, *e.g.* :

```
features
(
    {
        file "features.eMesh"; // file containing edge mesh
        level 2;               // level of refinement
    }
);
```

The *edgeMesh* containing the features can be extracted from the tri-surface file using *surfaceFeatureExtract* which specifies the tri-surface and controls such as included angle through a *surfaceFeatureExtractDict* configuration file, examples of which can be found in several tutorials and the *$FOAM_UTILITIES/surface/surfaceFeatureExtract* directory in the OpenFOAM installation. The utility is simply run by executing the following in a terminal

```
surfaceFeatureExtract
```

Following feature refinement, cells are selected for splitting in the locality of specified surfaces as illustrated in Figure 5.12.

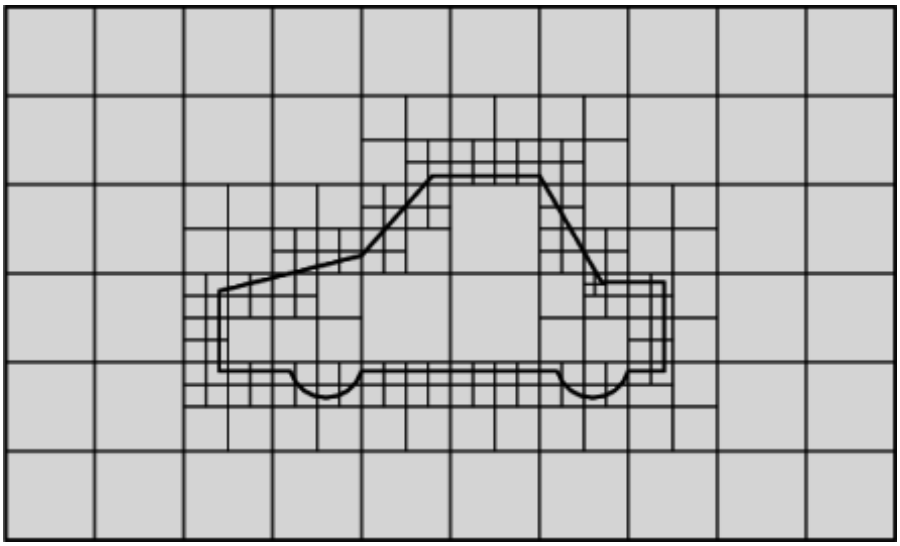

Figure 5.12: Cell splitting by surface in *snappyHexMesh* meshing process

The `refinementSurfaces` dictionary in *castellatedMeshControls* requires dictionary entries for each STL surface and a default `level` specification of the minimum and maximum refinement in the form `(<min> <max>)`. The minimum level is applied generally across the surface; the maximum level is applied to cells that can see intersections that form an angle in excess of that specified by `resolveFeatureAngle`.

The refinement can optionally be overridden on one or more specific region of an STL surface. The region entries are collected in a `regions` sub-dictionary. The keyword for each region entry is the name of the region itself and the refinement level is contained within a further sub-dictionary. An example is given below:

```
  refinementSurfaces
  {
      sphere.stl
      {
          level (2 2); // default (min max) refinement for whole surf
ace
          regions
          {
              secondSolid
              {
                  level (3 3); // optional refinement for secondSolid
 region
              }
          }
      }
  }
```

5.4.4 Cell removal

Once the feature and surface splitting is complete a process of cell removal begins. Cell removal requires one or more regions enclosed entirely by a bounding surface within the domain. The region in which cells are retained are simply identified by a location vector within that region, specified by the `locationInMesh` keyword in *`castellatedMeshControls`*. Cells are retained if, approximately speaking, 50% or more of their volume lies within the region. The remaining cells are removed accordingly as illustrated in Figure 5.13.

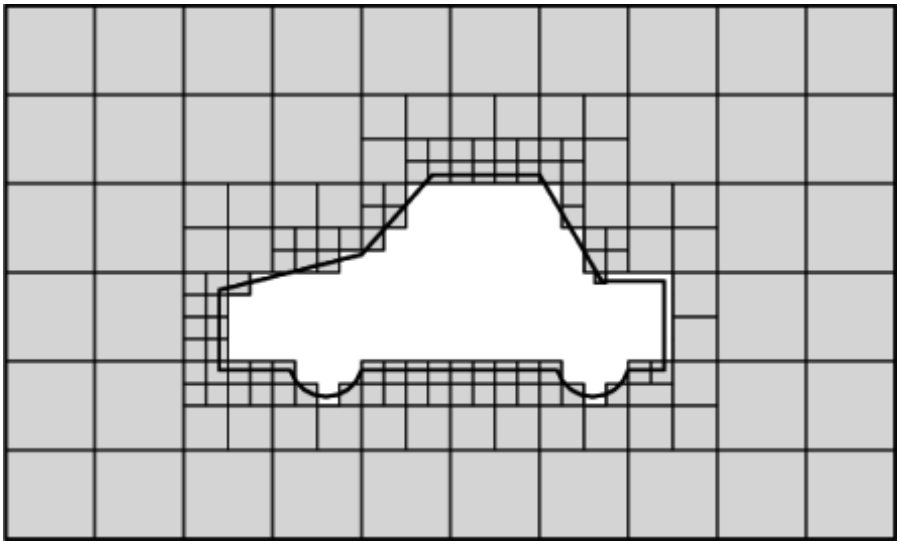

Figure 5.13: Cell removal in *`snappyHexMesh`* meshing process

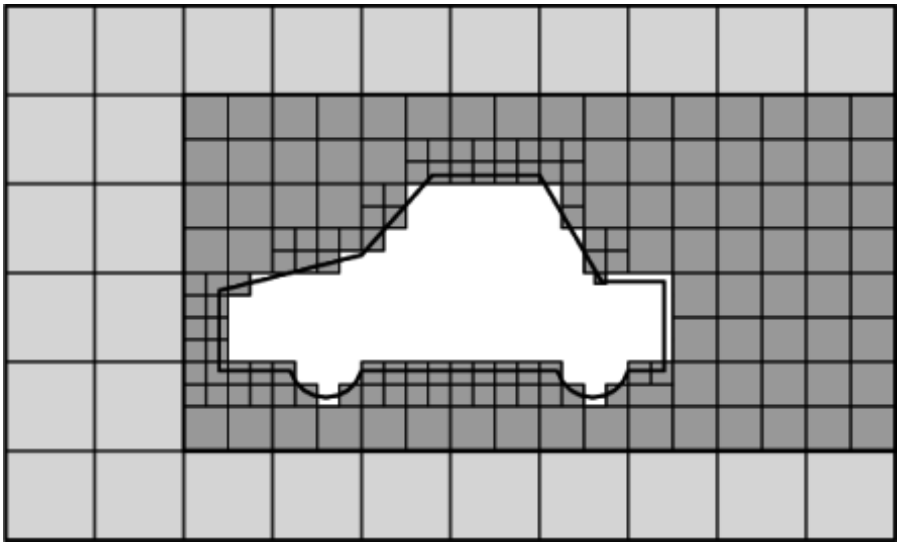

Figure 5.14: Cell splitting by region in *`snappyHexMesh`* meshing process

5.4.5 Cell splitting in specified regions

Those cells that lie within one or more specified volume regions can be further split as illustrated in Figure 5.14 by a rectangular region shown by dark shading. The `refinementRegions` sub-dictionary in *`castellatedMeshControls`* contains entries for refinement of the volume regions specified in the *`geometry`* sub-dictionary. A refinement `mode` is applied to each region which can be:

- `inside` refines inside the volume region;
- `outside` refines outside the volume region
- `distance` refines according to distance to the surface; and can accommodate different levels at multiple distances with the `levels` keyword.

For the `refinementRegions`, the refinement level is specified by the `levels` list of entries with the format `(<distance> <level>)`. In the case of `inside` and `outside` refinement, the `<distance>` is not required so is ignored (but it must be specified). Examples are shown below:

```
  refinementRegions
  {
      box1x1x1
      {
          mode inside;
          levels ((1.0 4));         // refinement level 4 (1.0 entry
ignored)
      }

      sphere.stl
      {                             // refinement level 5 within 1.0 m
          mode distance;            // refinement level 3 within 2.0 m
          levels ((1.0 5) (2.0 3)); // levels must be ordered nearest
 first
      }
  }
```

5.4.6 Snapping to surfaces

The next stage of the meshing process involves moving cell vertex points onto surface geometry to remove the jagged castellated surface from the mesh. The process is:

1. displace the vertices in the castellated boundary onto the STL surface;
2. solve for relaxation of the internal mesh with the latest displaced boundary vertices;

3. find the vertices that cause mesh quality parameters to be violated;
4. reduce the displacement of those vertices from their initial value (at 1) and repeat from 2 until mesh quality is satisfied.

The method uses the settings in the *snapControls* sub-dictionary in *snappyHexMeshDict*, listed in Table 5.8.

Keyword	Description	Example
nSmoothPatch	Number of patch smoothing iterations before finding correspondence to surface	3
tolerance	Ratio of distance for points to be attracted by surface feature point or edge, to local maximum edge length	4.0
nSolveIter	Number of mesh displacement relaxation iterations	30
nRelaxIter	Maximum number of snapping relaxation iterations	5

Table 5.8: Keywords in the *snapControls* dictionary of *snappyHexMeshDict*.

An example is illustrated in the schematic in Figure 5.15 (albeit with mesh motion that looks slightly unrealistic).

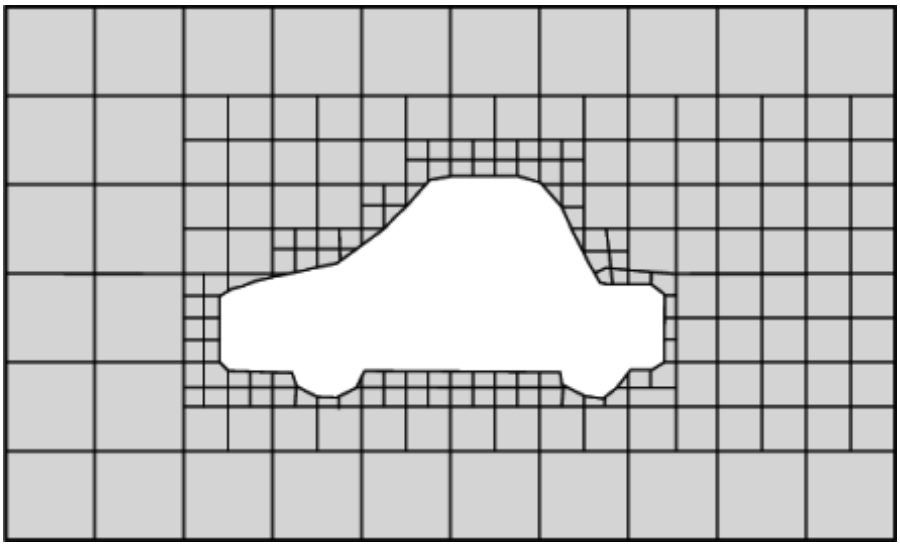

Figure 5.15: Surface snapping in *snappyHexMesh* meshing process

5.4.7 Mesh layers

The mesh output from the snapping stage may be suitable for the purpose, although it can produce some irregular cells along boundary surfaces. There is an optional stage of the meshing process which introduces additional layers of hexahedral cells aligned to the boundary surface as illustrated by the dark shaded cells in Figure 5.16.

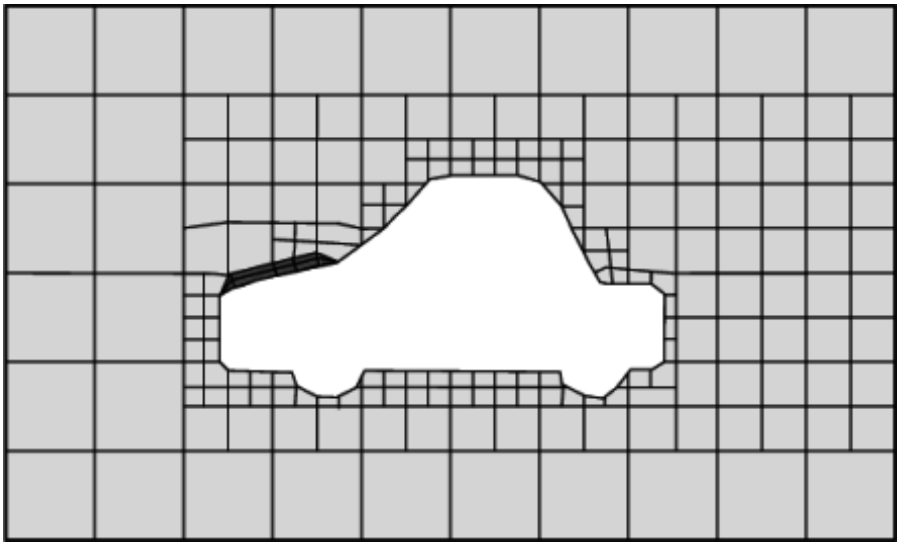

Figure 5.16: Layer addition in *snappyHexMesh* meshing process

The process of mesh layer addition involves shrinking the existing mesh from the boundary and inserting layers of cells, broadly as follows:

1. the mesh is projected back from the surface by a specified thickness in the direction normal to the surface;
2. solve for relaxation of the internal mesh with the latest projected boundary vertices;
3. check if validation criteria are satisfied otherwise reduce the projected thickness and return to 2; if validation cannot be satisfied for any thickness, do not insert layers;
4. if the validation criteria can be satisfied, insert mesh layers;
5. the mesh is checked again; if the checks fail, layers are removed and we return to 2.

The layer addition procedure uses the settings in the *addLayersControls* sub-dictionary in *snappyHexMeshDict*; entries are listed in Table 5.9.

Keyword	Description	Example
layers	Dictionary of layers	
relativeSizes	Are layer thicknesses relative to undistorted cell size outside	true/false

	layer or absolute?	
`expansionRatio`	Expansion factor for layer mesh	`1.0`
`finalLayerThickness`	Thickness of layer furthest from the wall, either relative or absolute according to the `relativeSizes` entry	`0.3`
`minThickness`	Minimum thickness of cell layer, either relative or absolute (as above)	`0.25`
`nGrow`	Number of layers of connected faces that are not grown if points get not extruded; helps convergence of layer addition close to features	`1`
`featureAngle`	Angle above which surface is not extruded	`60`
`nRelaxIter`	Maximum number of snapping relaxation iterations	`5`
`nSmoothSurfaceNormals`	Number of smoothing iterations of surface normals	`1`
`nSmoothNormals`	Number of smoothing iterations of interior mesh movement direction	`3`
`nSmoothThickness`	Smooth layer thickness over surface patches	`10`
`maxFaceThicknessRatio`	Stop layer growth on highly warped cells	`0.5`
`maxThicknessToMedialRatio`	Reduce layer growth where ratio thickness to medial distance is large	`0.3`
`minMedianAxisAngle`	Angle used to pick up medial axis points	`130`
`nBufferCellsNoExtrude`	Create buffer region for new layer terminations	`0`

nLayerIter	Overall max number of layer addition iterations	50
nRelaxedIter	Max number of iterations after which the controls in the *relaxed* sub dictionary of meshQuality are used	20

Table 5.9: Keywords in the *addLayersControls* sub-dictionary of *snappyHexMeshDict*.

The layers sub-dictionary contains entries for each *patch* on which the layers are to be applied and the number of surface layers required. The patch name is used because the layers addition relates to the existing mesh, not the surface geometry; hence applied to a patch, not a surface region. An example layers entry is as follows:

```
  layers
  {
      sphere.stl_firstSolid
      {
          nSurfaceLayers 1;
      }
      maxY
      {
          nSurfaceLayers 1;
      }
  }
```

Keyword	Description	Example
maxNonOrtho	Maximum non-orthogonality allowed; 180 disables	65
maxBoundarySkewness	Max boundary face skewness allowed; <0 disables	20
maxInternalSkewness	Max internal face skewness allowed; <0 disables	4
maxConcave	Max concaveness allowed; 180 disables	80

`minFlatness`	Ratio of minimum projected area to actual area; `-1` disables	`0.5`
`minVol`	Minimum pyramid volume; large negative number, *`e.g.`* `-1e30` disables	`1e-13`
`minArea`	Minimum face area; $<$`0` disables	`-1`
`minTwist`	Minimum face twist; $<$`-1` disables	`0.05`
`minDeterminant`	Minimum normalised cell determinant; `1` $=$ hex; $\leq$0 illegal cell	`0.001`
`minFaceWeight`	`0`$\rightarrow$`0.5`	`0.05`
`minVolRatio`	`0`$\rightarrow$`1.0`	`0.01`
`minTriangleTwist`	$>$`0` for *`Fluent`* compatability	`-1`
`nSmoothScale`	Number of error distribution iterations	`4`
`errorReduction`	Amount to scale back displacement at error points	`0.75`

`relaxed`	Sub-dictionary that can include modified values for the above keyword entries to be used when `nRelaxedIter` is exceeded in the layer addition process	`relaxed`
		{
		...
		}

Table 5.10: Keywords in the *`meshQualityControls`* sub-dictionary of *`snappyHexMeshDict`*.

5.4.8 Mesh quality controls

The mesh quality is controlled by the entries in the *`meshQualityControls`* sub-dictionary in *`snappyHexMeshDict`*; entries are listed in Table 5.10.

[prev] [next]

Chris Greenshields 1st March 2015 User Guide

← OpenFOAM User Guide: 5.3 Mesh generation with blockMesh

OpenFOAM User Guide: 5.5 Mesh conversion →