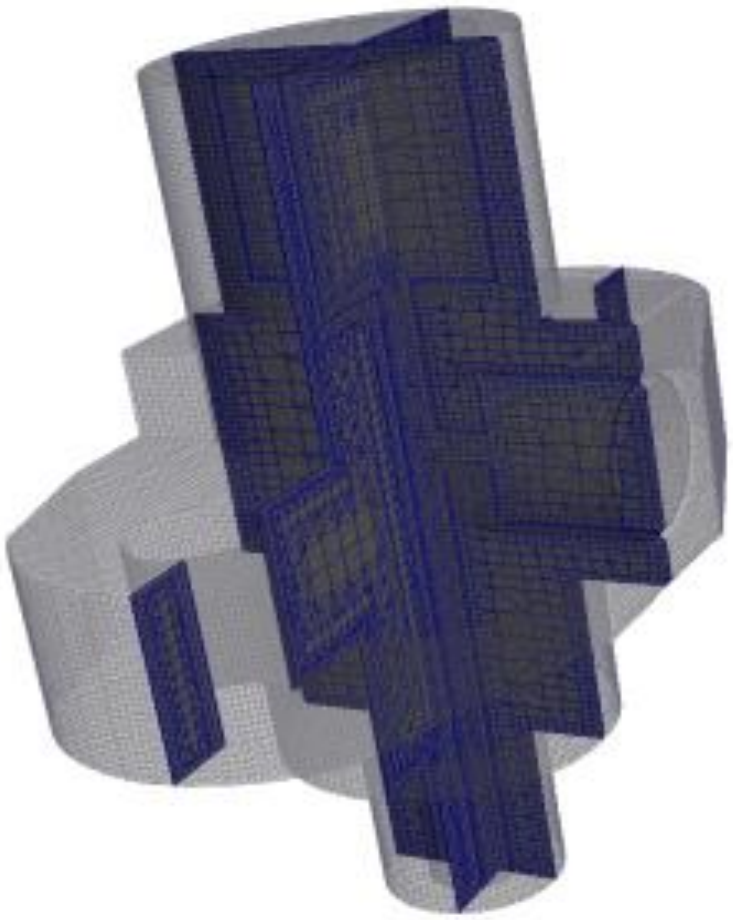
Tutorial Twelve

snappyHexMesh – Single Region



6th edition, April 2023

ENGINEERING FLUID DYNAMICS

Editorial board:

- Bahram Haddadi
- Christian Jordan
- Michael Harasek

Compatibility:

- OpenFOAM® v10

Cover picture from:

- Philipp Schretter

Contributors:

- Philipp Schretter
- Bahram Haddadi
- Yitong Chen

TU WIEN ICEBE IMAGINEERING NATURE

ISBN 978-3-903337-02-2

Publisher: chemical-engineering.at

Available from: www.fluiddynamics.at

Background

In this tutorial, we will familiarize ourselves with the *snappyHexMesh* tool in OpenFOAM®. This utility generates 3D meshes containing hexahedra and split-hexahedra. We will also introduce different types of meshes with complex geometries and compare the *snappyHexMesh* tool with other mesh generation tools.

1. Meshes with complex geometries

So far we have only worked with meshes in Cartesian co-ordinates, however, many engineering problems involve complex geometries that do not fit exactly in Cartesian co-ordinates. In such cases, it would be much more advantageous to work with grids that can handle curvature and geometric complexity more naturally.

CFD methods for complex geometries are classified into two groups:

1. structured curvilinear grid arrangements
2. unstructured grid arrangements

In a **structured grid** arrangements:

- Cells center points are placed at the intersections of co-ordinates lines
- Cells have a fixed number of neighboring cells
- Cells center points can be mapped into a matrix based on their location in the grid
- Structure and position in the matrix is given by indices (*I*, *J* in two dimensions and *I*, *J*, *K* in three dimensions)

For the most complex geometries it may be necessary to sub-divide the flow domain into several different blocks, where each mesh cell is a block, this is known as **block-structured grids**. The next level of complexity is the **unstructured grids**. It gives unlimited geometric flexibility, here the limitations of structured grids do not apply – but at the cost of higher programming and computational efforts. Unstructured grids also allow the most efficient use of computing resources for complex flows, so this technique is now widely used in industrial CFD.

2. Mesh generation tools

There are a number of advanced meshing tools available, both commercial and free source. The major mesh generators are ANSYS GAMBIT®, ICEM, Salome, snappyHexMesh and cfMesh. Here we will learn about GAMBIT®, snappyHexMesh and cfMesh tools in detail.

2.1. GAMBIT®

GAMBIT® is a 3D unstructured tool, to specify the meshing scheme in it, two parameters must be specified:

- Elements
- Type

The Elements parameter defines the shape(s) of the elements that are used to mesh the object. The Type parameter defines the pattern of mesh elements on the object. It has a single graphical user interface which brings geometry creation and meshing together in one environment.

2.2. snappyHexMesh

In contrast to GAMBIT®, which incorporates both mesh generation and refinement, the *snappyHexMesh* tool built within OpenFOAM® requires an existing geometry base mesh to work with. The base mesh usually comes from the blockMesh tool. This utility has the following key features:

- allow parallel execution to speed up the process
- supports geometry data from STL/OBJ files
- addition of internal and wall layers
- zonal meshing

The key steps involved when running snappyHexMesh are:

- **Castellation**: The cells which are beyond a region set by a predefined point are deleted
- **Snapping**: Reconstructs the cells to move the edges from inside the region to the required boundary
- **Layering**: Creates additional layers in the boundary region.

The advantages of snappyHexMesh over the other mesh generation tools are as follows:

- No commercial software package is ultimately necessary. For the meshing, the OpenFOAM® environment is sufficient and no further software is necessary.
- The geometry can be created with any CAD program like CATIA®, FreeCAD, etc. As the geometry is to be only surface data, the files need to be in .stl, .nas or .obj. format.
- The meshing process can be run in parallel mode. If high computational capabilities are available, high quality meshes can be generated in little time.

2.3. cfMesh

cfMesh is an open-source library for mesh generation implemented within the OpenFOAM® framework (similar to *snappyHexMesh*). Currently cfMesh is capable of producing mesh of Cartesian type in both 2D and 3D, tetrahedral and polyhedral types.

The fundamental work-flow of the tool starts from a mesh template, then followed by a mesh modifier. The modifier allows for efficient parallelization using shared memory parallelization (SMP) and distributed memory parallelization using MPI.

snappyHexMesh – flange

Tutorial outline

The procedure described in this tutorial is structured in the following order:

- Creation of the geometry data
- Meshing a geometry with one single region
- Run an OpenFOAM® simulation with the generated mesh using scalarTransportFoam

Objectives

- The aim of the tutorial is to introduce single region meshing with the meshing tool snappyHexMesh
- Understanding the advantages of snappyHexMesh
- Understanding the three basic steps of snappyHexMesh

Data processing

Import your simulation to ParaView and analyze the heat distribution in the flange.

1. Pre-processing

1.1. Copying tutorial

Copy the following tutorial to your working directory.

```
$FOAM_TUTORIALS/mesh/snappyHexMesh/flange
```

1.2. Set-up of stl files

Normally the .stl files are created using CAD software, such as CATIA® and freeCAD. stl files contain information about the solid geometry. However, in this tutorial the stl files are available to be copied from the OpenFOAM® tutorials folder. To do this, copy the stl files from the below location to the constant/geometry of your running case directory.

```
$FOAM_TUTORIALS/resources/geometry/flange.stl.gz
```

1.3. constant directory

The *constant* directory must initially have the following folder:

- geoemtry*:* The folder **geometry** should contain a file with the geometry data to be meshed (stl, nas, obj). The file name is to be used as a reference pointer in later stages.

Note: The stl file should be in ascii format. All the stl files (different boundaries stl files) should form a closed geometry together.

1.4. system directory

For creating a mesh using snappyHexMesh, the following files should be present in system directory:

- **blockMeshDict**: For meshing using snappyHexMesh a background mesh is needed, which should surround the geometry surface (e.g. stl file) file. The background mesh will be refined based on the settings in the snappyHexMeshDict and the extra parts will be removed. Usually the background mesh is crated using blockMesh. Here we define a base mesh.

Note: To ensure that the sharp edges are refined properly, it is very important to create perfect cube cells in the background mesh using blockMesh utility.

- **decomposeParDict***:* The meshing using snappyHexMesh can be also performed in parallel mode. If the mesh is to be run in parallel using the *decomposePar* utility, this file defines the parameters for distributed processors
- **meshQualityDict***:* Parameters to be checked for mesh quality and their values are defined in this file.
- **surfaceFeatureExtractDict**: Using surfaceFeatures utility prior to meshing with snappyHexMesh helps to extract the sharp edges and have a better mesh

with snappyHexMesh on these edges. All edges are marked, whose adjacent surfaces normal are at an angle less than the angle specified in `includedAngle` in the *surfaceFeaturesDict*. The extracted edges are written to "*.extendedFeatureEdgeMesh" files in constant/extendedFeatureEdgeMesh folder to be treated later in the meshing process.

```
   //  * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * //
Surfaces ("flange.stl");

includedAngle    150;
   //  * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * //
```

- **snappyHexMeshDict**: This file includes the settings for running the snappyHexMesh. As mentioned in the Background section meshing using this tool has three steps:
  1) Castellating
  2) Snapping
  3) Layering

In the first section of this file, `castellatedMesh`, `snap`, `addLayers` can be set to true or false depending on the stages required. In the following setting, constellating and snapping are active and adding layers is deactivated (to activate it, set the flag to true).

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

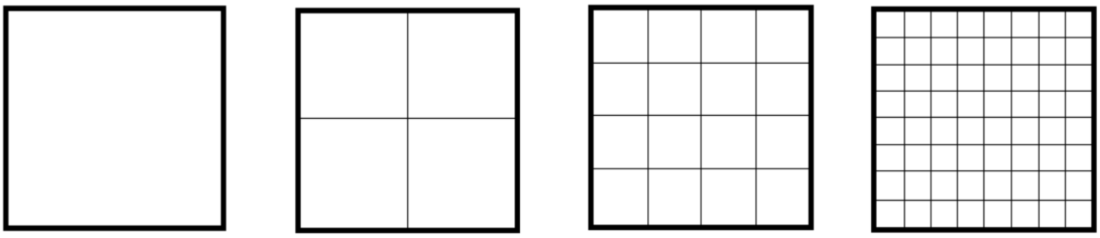
The `Geometry` sub-dictionary lists all surfaces used by *snappyHexMeshDict*, except the blockMesh geometry, and defines a name for each of them to be used as a reference.

Then we specify a region of the domain that we want to refine. The refined region is given an arbitrary name; in this case, it is `refineHole`, which is a sphere with its center and radius defined.

```
//  * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * //
geometry
{
    flange
    {
        type triSurfaceMesh;
        file "flange.stl";
    }
    refineHole
{
    type searchableSphere;
    centre (0 0 -0.012);
    radius 0.003;
    }
};
   //  * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * //
```

1.4.1. CASTELLATING

In the castellating step based on the settings in the *snappyHexMeshDict* file the created background mesh (in this case using blockMesh) cells are cut into sub-cells and the unneeded part of the mesh will be deleted. The background mesh is known as mesh “level 0”, by setting the “level” to 1 the background mesh at the position of features or defined refinements will be cut into half in each direction (creating 8 sub-cells for a 3D mesh). Therefor by each level of refinement number of cells increases by factor 8!



Refinement level 0, level 1, level 2, level 3

The `castellatedMeshControls` sub-dictionary is used for user-defined mesh refinement in the castellating step.

`features` allows special treatment of the “*.extendedFeatureEdgeMesh” edges to be refined to a certain level.

`refinementSurfaces` are for surface based refinement. Every surface is specified with two levels. The first level is the minimum level that every cell intersecting the surface gets refined up to. The second level is the maximum level of refinement. If the type of thesurface is also defined (e.g. patch or wall) the surface will be marked as a boundary with the assigned name.

`resolveFeatureAngle` is an important setting. Edges, whose adjacent surfaces normal are at an angle higher than the value set, are resolved. The lower the value, the better the resolution at sharp edges.

`refinementRegions`: Volume based refinement of the regions defined in the geometry section. In this tutorial the `refinementHole` region will be refined. In the levels the first number (1E15) is the maximum number of the cells which can be reached after refinement in this region and second number (3) is the level of refinement

`locationInMesh`: Important coordinate for single region cases, to define which part of the mesh should be kept, inside or outside the geometry.

```
   //  * * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * //
castellatedMeshControls
{
    maxLocalCells 100000;
    maxGlobalCells 2000000;
    minRefinementCells 0;
nCellsBetweenLevels 1;


    features
    (
        {
            file "flange.extendedFeatureEdgeMesh";
            level 0;
```

```
        }
    );

    refinementSurfaces
    {
        flange
        {
                level (2 2);
        }
    }

    resolveFeatureAngle 30;

    refinementRegions
{
    refineHole
    {
       mode inside;
       levels ((1E15 3));
    }
          locationInMesh (-9.23149e-05 -0.0025 -0.0025);
    allowFreeStandingZoneFaces true;
}
   //  * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * //
```

Note: The `locationInMesh` point should never be on a face of the mesh, even after refinement. It should always be inside a cell or the meshing will fail!

In the castellated step, the background mesh will be refined based on the defined refinement levels at features, surfaces or regions and the unneeded part of the mesh will be removed.

1.4.2. SNAPPING

Important parameters are number of mesh displacement iterations, `nSolveIter` and the number of feature edge snapping iterations, `nFeatureSnapIter`.

```
   //  * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * //
snapControls
{
    nSmoothPatch 3;
    tolerance 1.0;
    nSolveIter 300;
    nRelaxIter 5;
        nFeatureSnapIter 10;
        implicitFeatureSnap false;
        explicitFeatureSnap true;
        multiRegionFeatureSnap true;
}
   //  * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * //
```

1.4.3. LAYERING

The label for the layering is equal to the labeling of the Boundary surface in the boundary file in the constant/polyMesh folder.

- `nSurfaceLayers` defines the number of surface layers
- `expansionRatio` defines the expansion ratio of the surface layers
- `finalLayerThickness` and `minThickness` define the min and the final thickness of the surface layers

- `nLayerIter`: if not snapped smoothly enough, the max number of layer addition iteration can be increased.

```
   //  * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * //
addLayersControls
{
    relativeSizes true;
    layers
    {
        "flange_.*"
        {
            nSurfaceLayers  3;
        }
    }
expansionRatio 1.005;

    finalLayerThickness 0.3;
    minThickness 0.25;
    nGrow 0;
    featureAngle 30;
    nRelaxIter 5;
    nSmoothSurfaceNormals 1;
    nSmoothNormals 3;
    nSmoothThickness 10;
    maxFaceThicknessRatio 0.5;
    maxThicknessToMedialRatio 0.3;
    minMedianAxisAngle 90;
    nBufferCellsNoExtrude 0;
nLayerIter 50;
nRelaxedIter 20;
}
meshQualityControls
{
    #include "meshQualityDict"
relaxed
{
    maxNonOrtho 75;
}

nSmoothScale 4;
    errorReduction 0.75;
}
writeFlags
(
    scalarLevels
    layerSets
    layerFields
);
mergeTolerance 1e-6;
   //  * * * * * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * //
```

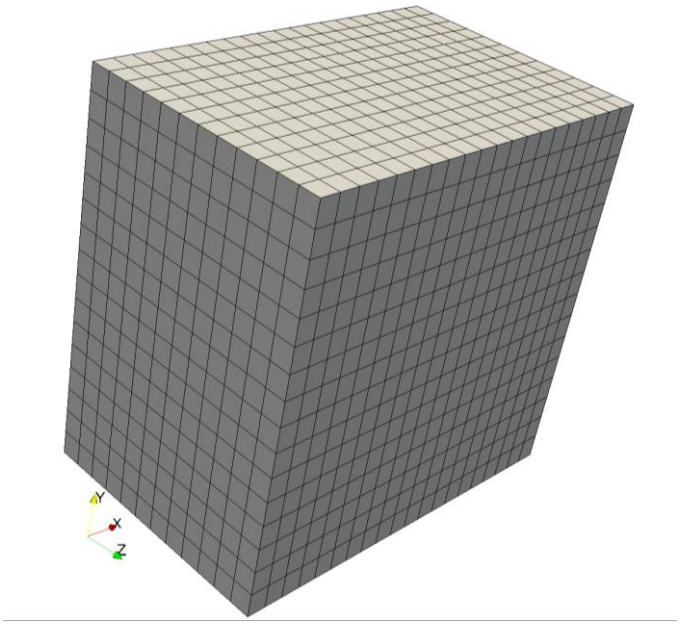
Note: Only the relevant changes, which were used in the sample flange case, are commented in the snappyHexMeshDict.

2. Running snappyHexMesh

The background mesh is created with the following command:

```
>blockMesh
```

According to the settings in the *blockMeshDict*, the mesh was created with 20 cells in x direction, 16 cells in y direction and with 12 cells in z direction.



Block mesh for flange

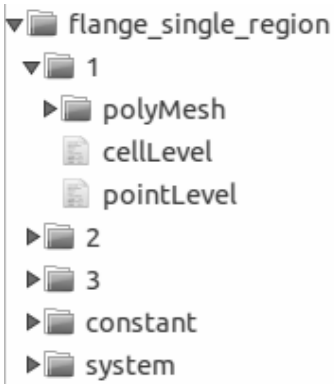
```
>surfaceFeatures
```

The command to mesh the flange geometry on one processor is

```
>snappyHexMesh
```

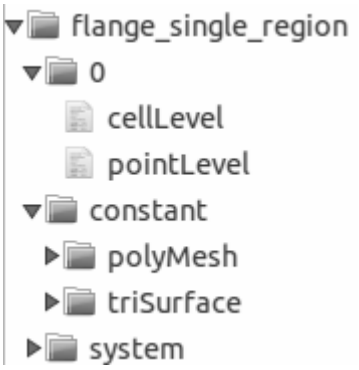
Note: The meshing process with snappyHexMesh can also be run in parallel. To run the command on several processors, refer to Tutorial Eight for more information.

The command `snappyHexMesh` creates a folder with the mesh files for each mesh step. If, for example, in the **snappyHexMeshDict**, only *castellatedMesh* is set to *true* and *snap* and *addLayers* are set to *false*, only one folder is created. If also *snap* is set to *true*, 2 folders are created and if also *addLayers* is set to *true*, 3 folders with 3 polyMesh folders are created.



Folders structure after running snappyHexMesh

In order to avoid the creation of these folders and only keep the final mesh, the following command can be used to overwrite the previous meshing steps. In this case, only one polyMesh folder exits in the /constant directory.



Folders structure after using -overwrite flag

```
>snappyHexMesh -overwrite
```

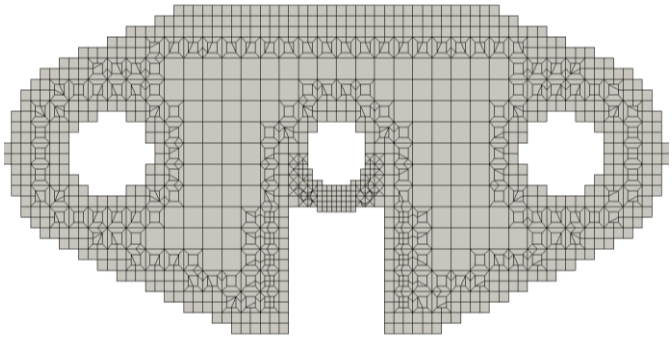
However, sometimes it is useful to run snappyHexMesh without the overwrite option, as it allows the user to make changes to a specific time step without having to run all the other steps again, thus reducing computational time.

3. Examining the meshes

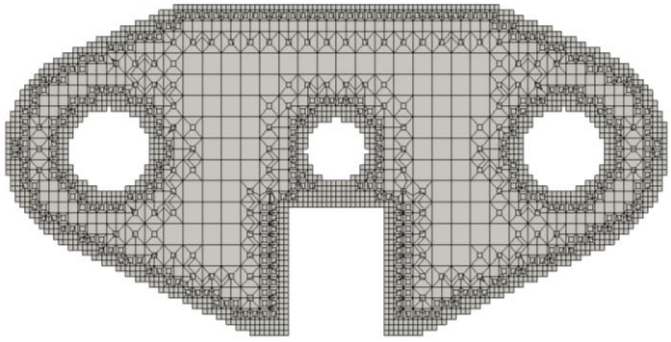
To examine, what each of the steps in the *snappyHexMeshDict* really does, we need to turn off the overwrite feature in snappyHexMesh command and generate VTK files to be opened in ParaView.

```
>foamToVTK
```

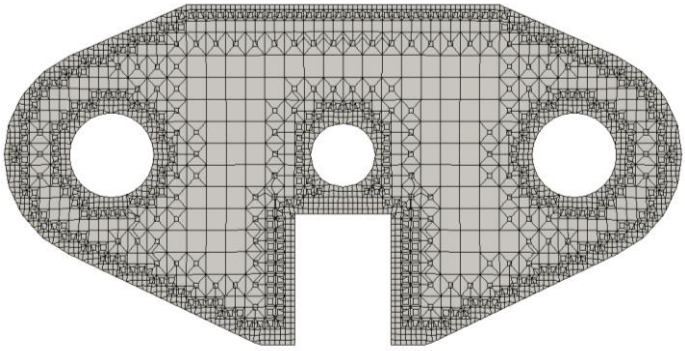
Simply change the time in Paraview to see the effect of *snappyHexMesh* steps on the mesh, i.e. time 1 corresponds to the mesh after castellating step, time 2 for the mesh after snapping step, time 3 for the mesh after the layering step.



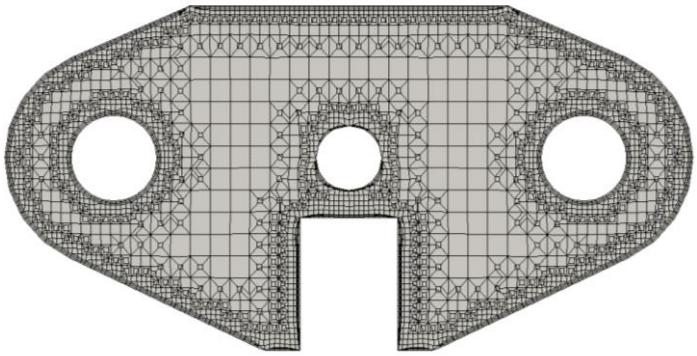
Flange mesh for step castellating with surface refinement level 2



Flange mesh for step castellating with surface refinement level 3

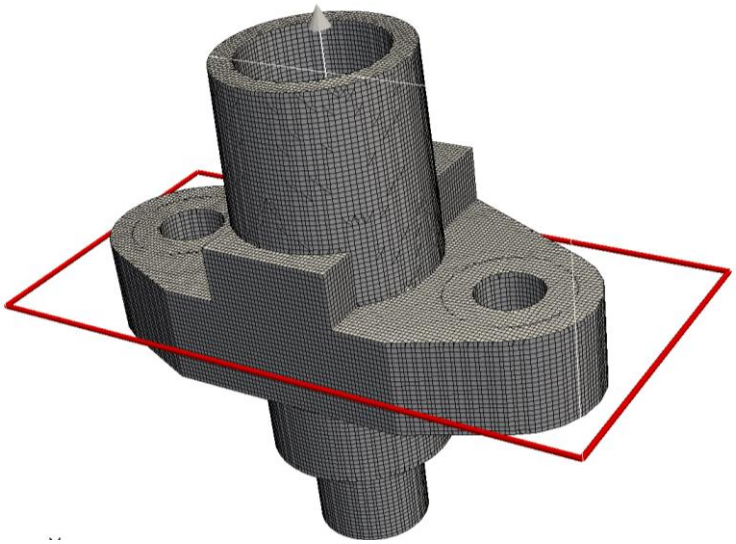


Flange mesh for step snap with surface refinement level 3



Flange mesh for step addlayers with surface refinement level 3

The slice views taken with ParaView from the center of the flange. The slices are depicted by the red plain in the following figure:



Flange with sectional plain

You can review the mesh quality with the tool *checkMesh*.

```
>checkMesh
```

4. Running simulation

4.1. Copy tutorial

Now with the new mesh ready, let's run some simulation on it! Here scalarTransportFoam solver is chosen for the simulation. To set up the case, copy the following tutorial file into your working directory:

```
$FOAM_TUTORIALS/basic/scalarTransportFoam/pitzDaily
```

The flange mesh files need to be transferred to the running case directory. To achieve this, copy the polyMesh folder from the latest time step file of the flange folder into the constant directory of the pitzDaily folder. If the overwrite function is activated when running snappyHexMesh, copy the polyMesh folder from constant directory of the flange folder.

4.2. Case set-up

The following changes need to be made to set up the case:

- Update the T file in the 0 directory, so that the flange has an initial temperature of 293K but is heated up from the inlet at 350K

```
dimensions      [0 0 0 1 0 0 0];

internalField   uniform 293;

boundaryField
{
   flange_patch1
    {
        type            fixedValue;
        value           uniform 350;
    }
```

```
    flange_patch2
    {
        type            fixedValue;
        value           uniform 293;
    }
    flange_patch3
    {
        type            fixedValue;
        value           uniform 293;
    }
    flange_patch4
    {
        type            fixedValue;
        value           uniform 350;
    }
}
```

- Update the U file in the 0 directory so that the velocity in the entire flange domain and at the boundaries is zero
- Update the *controlDict* file in the system directory by changing the `endTime` to 0.0005, `deltaT` to 0.000001 and `writeInterval` to 100.

4.3. Running solver

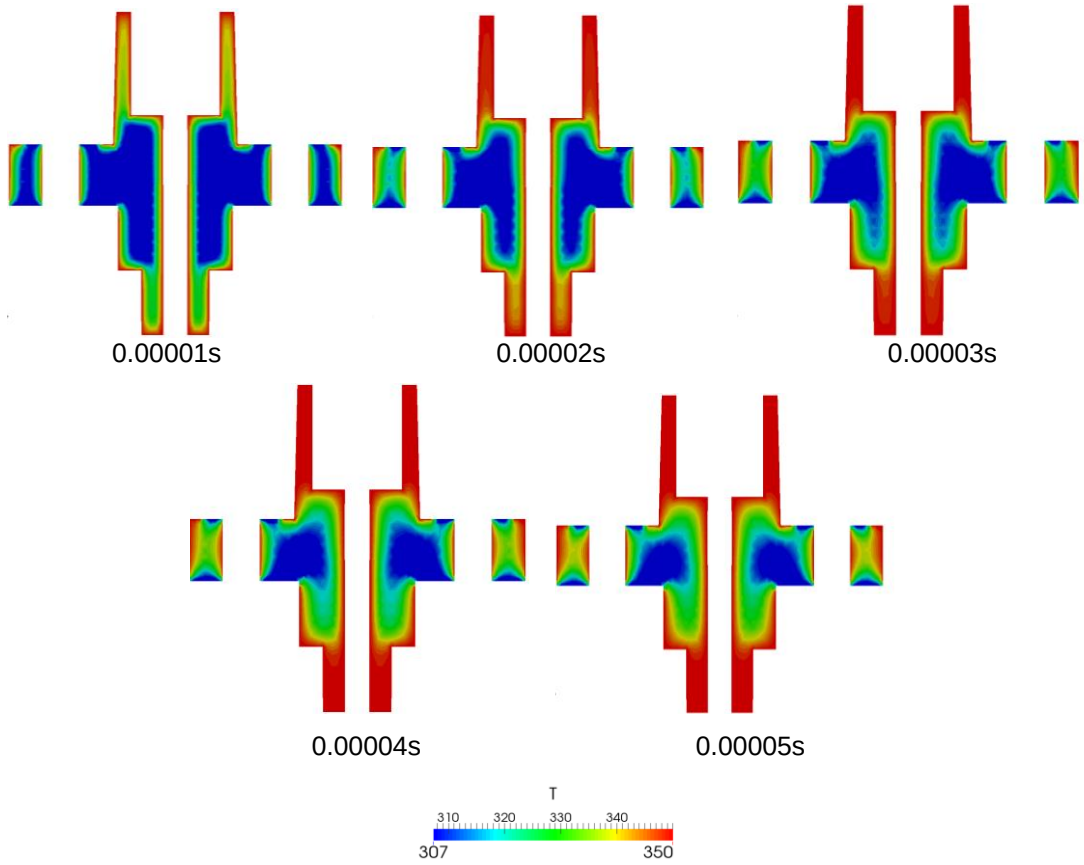
Run the solver with the command

```
>scalarTransportFoam
```

4.4. Results

Convert the results to VTK files with

```
>foamToVTK
```



Heating of the flange from 0.01 to 0.05s

ISBN 978-3-903337-02-2