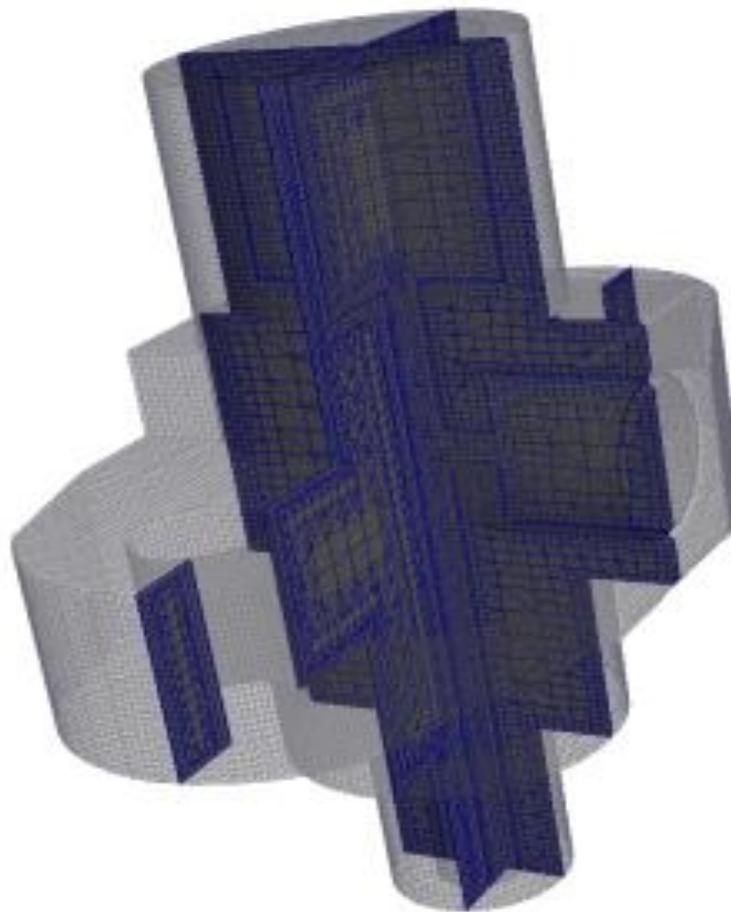


Tutorial Twelve

snappyHexMesh – Single Region



4th edition, Jan. 2018



This offering is not approved or endorsed by ESI® Group, ESI-OpenCFD® or the OpenFOAM® Foundation, the producer of the OpenFOAM® software and owner of the OpenFOAM® trademark.



Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by-nc-sa/3.0/>

Editorial board:

- Bahram Haddadi
- Christian Jordan
- Michael Harasek

Contributors:

- Philipp Schretter
- Yitong Chen

Compatibility:

- OpenFOAM® 5.0
- OpenFOAM® v1712

Cover picture from:

- Philipp Schretter

 Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by-nc-sa/3.0/>

Attribution–NonCommercial–ShareAlike 3.0 Unported (CC BY–NC–SA 3.0)

This is a human-readable summary of the Legal Code (the full license).

Disclaimer

You are free:

- to Share — to copy, distribute and transmit the work
- to Remix — to adapt the work

Under the following conditions:

- Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- Noncommercial — You may not use this work for commercial purposes.
- Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

With the understanding that:

- Waiver — Any of the above conditions can be waived if you get permission from the copyright holder.
- Public Domain — Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.
- Other Rights — In no way are any of the following rights affected by the license:
- Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
- The author's moral rights;
- Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.
- Notice — For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.

For more tutorials visit: www.cfd.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

Simulation

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 give a basic introduction to 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. Analyze the heat distribution in the flange.

1. Pre-processing

1.1. Copy 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/triSurface 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:

- **triSurface:**

The folder **triSurface** 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 created 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.


```

        flange
        {
            level (2 2);
        }
    }

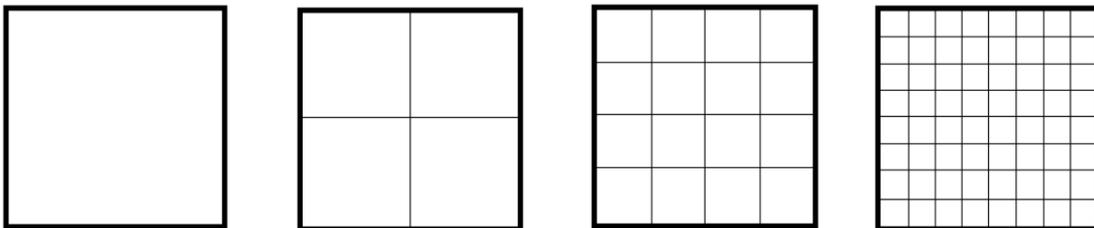
    resolveFeatureAngle 30;

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

```

In the first step, the `castellatedMeshControls`, the initial mesh can be refined with levels. The level depends on the refinement set in the `blockMesh`, and the required refinement on the surfaces. Therefore, levels can be set in the sub-directories `features`, `refinementSurfaces` and `refinementRegions` in the *snappyHexMeshDict*.

Level 0 stands for no refinement and each subsequent level splits the cell into 4 separate cells.



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

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;
}
// * * * * * //

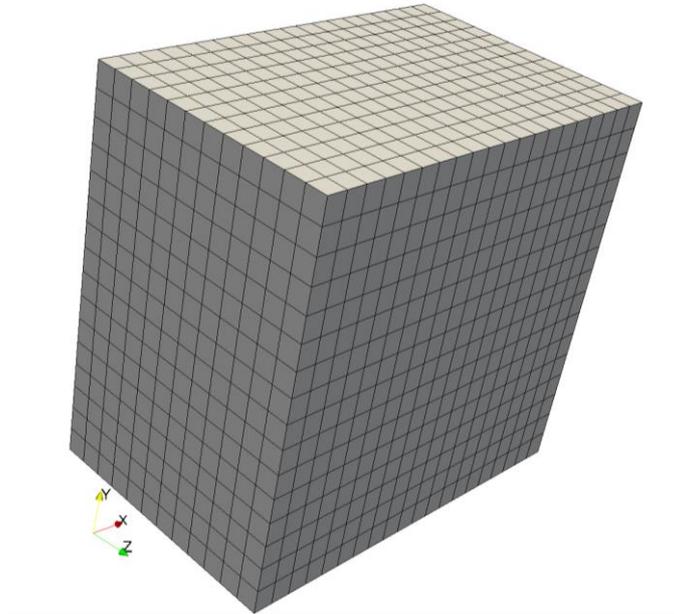
```

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

16 cells in y direction and with 12 cells in z direction.



Block mesh for flange

The command `surfaceFeatureExtract` creates the **eMesh** files from the stl files and creates the folder `extendedFeatureEdgeMesh` in the constant directory. Refining certain parts of the surface geometry with the `surfaceFeatureExtract` tool is optional.

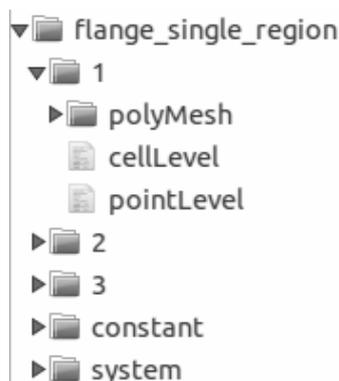
```
>surfaceFeatureExtract
```

The command to mesh the flange geometry on one processor is

```
>snappyHexMesh
```

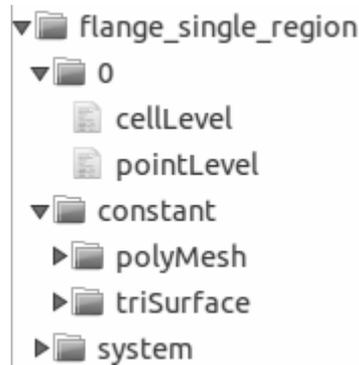
Note: The meshing process with `snappyHexMesh` can also be run in on several processors in parallel. To run the command on several processors, refer to Tutorial Eight for more information.

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 exists in the /constant directory.



Folders structure after using -overwrite flag

```
>snappyHexMesh -overwrite
```

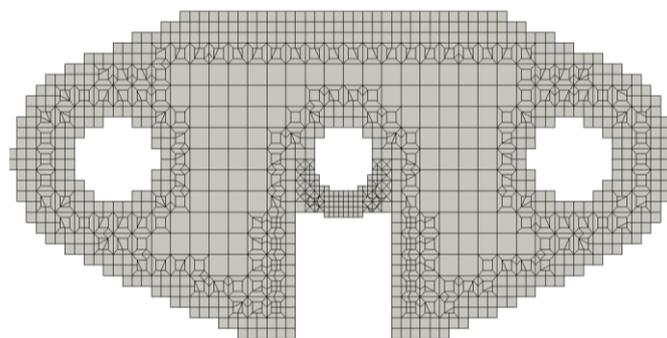
However, sometimes it is useful to run `snappyHexMesh` without the `overwrite` option, as it allows the user to make changes to a specific time step without having to run all the other steps again, thus reducing computational time.

3. Examining the meshes

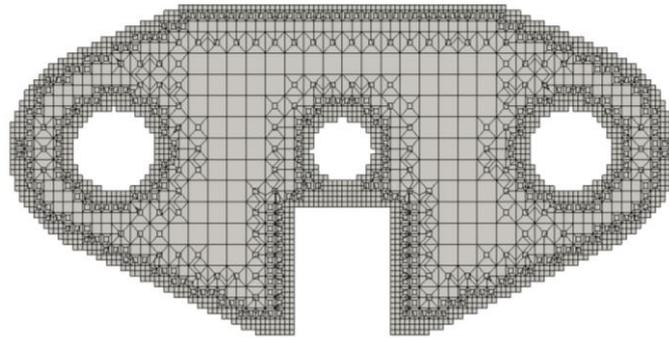
To examine, what each of the steps in the `snappyHexMeshDict` really does, we need to turn off the `overwrite` feature in `snappyHexMesh` command and generate VTK files to be opened in ParaView.

```
>foamToVTK
```

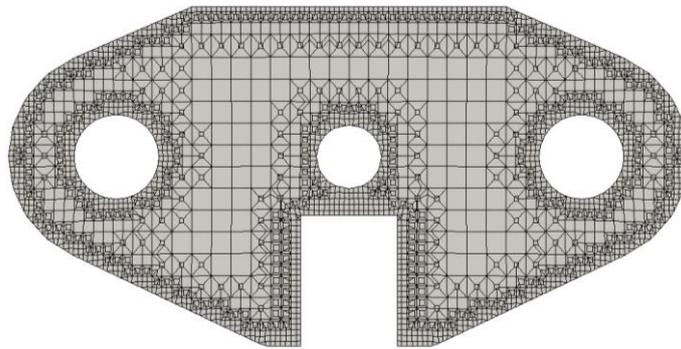
Simply change the time in Paraview to see the effect of `snappyHexMesh` steps on the mesh, i.e. time 1 corresponds to the mesh after castellating step, time 2 for the mesh after snapping step, time 3 for the mesh after the layering step.



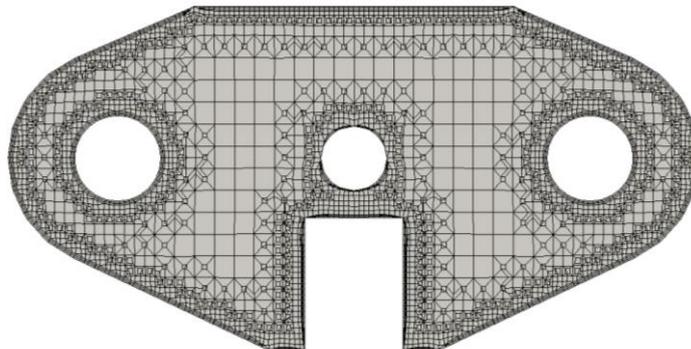
Flange mesh for step castellate with surface refinement level 2



Flange mesh for step castellate with surface refinement level 3

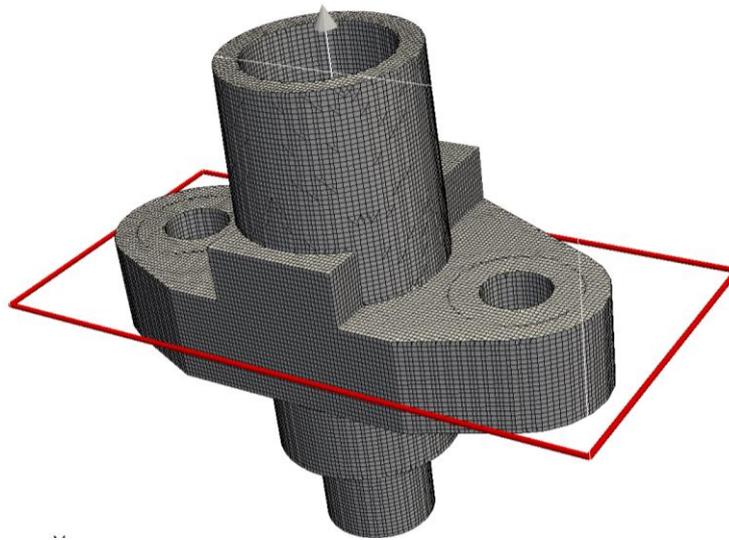


Flange mesh for step snap with surface refinement level 3



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

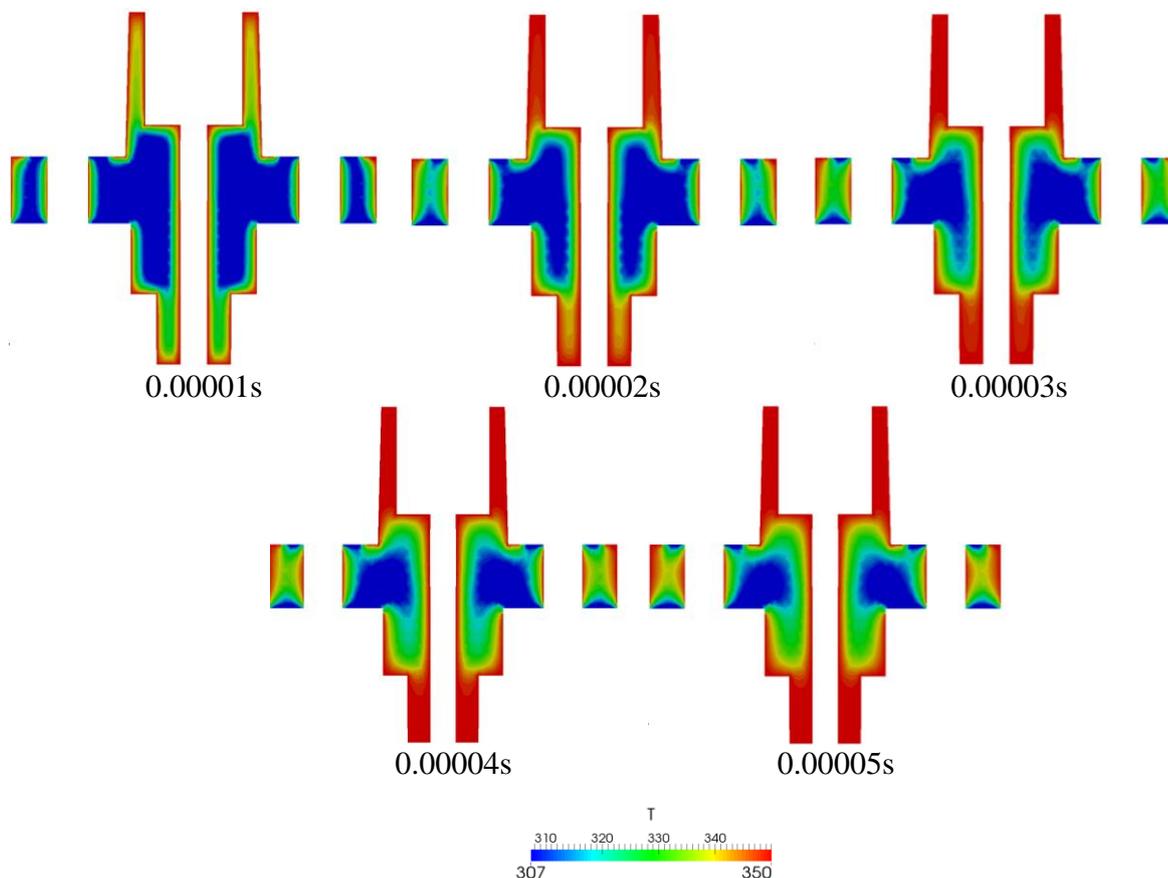
Run the solver with the command

```
>scalarTransportFoam
```

4.4. Results

Convert the results to VTK files with

```
>foamToVTK
```



Heating of the flange from 0.01 to 0.05s