Tutorial 2: Meshing with `snappyHexMesh`

Learning Outcomes:

Meshing with `snappyHexMesh`; further use of `paraFoam`, mesh transformation with `transformPoints`; `foamLog`

II.1 Introduction

Meshing is one of the most significant parts of CFD. The quality of the mesh can have a crucial impact on the accuracy of the results, or even determine whether a solution can be found. Creating the mesh is also the most human time-intensive part of the whole CFD process. For these reasons a wide range of tools have been developed to aid and improve the CFD meshing process. One important task is the automatic meshing of geometries taken from CAD. This is surprisingly difficult, because the quality of CAD output necessary to successfully run an automatic mesh generator is considerably higher than is necessary for other purposes. For instance, many complex CAD files contain numerous small faces which would not cause problems in normal circumstances, but which can cause an automatic meshing algorithm to fail.

OpenFOAM includes a highly sophisticated CAD-based automatic mesh generator called `snappyHexMesh` which does a good job of meshing complex geometries defined by .stl (stereolithography) files. It does this by taking a rectangular base mesh and truncating and manipulating the mesh where it intersects the .stl surface. Mesh refinement and boundary layer manipulation can also be specified as required. The output is a close approximation to the true shape of the object being meshed. These techniques are not 100% faithful to the geometry (unlike a tool such as Gambit or Pointwise which control the geometry as part of the meshing process) but are fast and can work directly with the common output from upstream CAD.

II.2 Meshing

This tutorial uses `snappyHexMesh` to mesh a well known aerospace vehicle. `snappyHexMesh` is a standard OpenFOAM code invoked in the usual manner and controlled via a dictionary `snappyHexMeshDict` in `system`. Due to the large number of options which control the behaviour of `snappyHexMesh`, this is quite a long file, but at this stage only a few options need be considered. The .stl file is stored in a subdirectory of `constant` called `triSurface`. `snappyHexMesh` reads this in, together with a rectangular block of cells usually generated using `blockMesh in the usual manner`. `snappyHexMesh` uses a 3-stage process, controlled by a series of flags at the start of the `snappyHexMeshDict` dictionary : `castellatedMesh`, `snap`, `addLayers`. These are the steps of truncating the base hexahedral mesh to conform approximately to the .stl file (including mesh refinement as required), snapping the near-wall vertices to lie on the surface, and adding near-wall boundary layer cells.

The tutorial directory contains 2 cases; shuttleSnappy, which will be used to do the meshing, and shuttleSmall which will run the case.

- Go into shuttleSnappy and run `blockMesh` to create the base mesh. It is informative to see the model, so start `paraFoam`; you will see the basic rectangular block (switch to `outline` for the display of this); File→Open, then browse to `constant/triSurface` to find the .stl file.
- Run `snappyHexMesh`. This will generate timestep directories `1 -- 3` representing the various stages of the meshing process.
- Examine the various steps of the simulation. There are two ways of visualising this. The first is to cut the geometry using either `clip` or `Extract Cells By Region` filters. The first cuts the geometry at a defined plane and throws away everything on one side, the other deletes all cells whose centres are on one side. The alternative is the `Extract Blocks` filter, which allows you to specify which topological entities are being shown. Make sure that all mesh parts in the base reader are selected (new boundaries appear for timestep 1) and that these filters are connected to the base reader, not to each other. What do you notice about the geometry?
- We can also look at the mesh statistics by running the OpenFOAM utility `checkMesh`. This will generate mesh statistics for each timestep and from this you can see how the quality changes for the different stages of the meshing.
- Change the `refinementSurfaces` settings from `level (2 4)` to `level (3 5)`, and re-run the meshing. This will take longer to run but should provide a much better mesh. What are the differences between the two meshes?

II.3 Simulation setup

Having generated a (hopefully) high quality mesh for this problem, we now need to set up a simulation to run. `snappyHexMesh` will run with an option `-overwrite` in which the mesh information overwrites the values in `constant`; however for this tutorial we will copy the mesh information into a new case. The original .stl file was specified in cm, with the result that the domain is over 1km in length. We will learn how to scale this, and also some valuable techniques for postprocessing the data.

First, copy the `polyMesh` directory from the timestep directory `3` to the correct location in `shuttleSmall` (i.e. `shuttleSmall/constant/polyMesh`). Run `checkMesh` and verify the domain size. The mesh can be scaled using the `transformPoints` utility :

```
transformPoints -scale "(0.01 0.01 0.01)"
```

`transformPoints` has a number of options allowing us to rotate, translate and otherwise modify the domain – in this case, scaling the domain by a factor of 0.01 in each direction. Run `checkMesh` again to check that the change has registered.

The `0.orig` timestep directory contains field files for the appropriate fields. Open these and check against the `boundaries` file in `polyMesh` that the boundaries are correctly specified. Note that the inlet turbulent conditions are specified using new boundary conditions, `turbulentIntensityKineticEnergyInlet` for the k field and `turbulentMixingLengthDissipationRateInlet` for the ϵ field, allowing us to directly specify the turbulent intensity and length scale, rather than having to work them out. The initial condition for velocity is a uniform velocity of $10m/s$ which can be improved by running the utility `potentialFoam` on the case. This solves the potential flow equations for this case. Copy the `0.orig` directory to `0` (`cp -r 0.orig 0`) and run `potentialFoam`, which will overwrite the appropriate files to create the initial conditions for the simulation.

The `controlDict` file contains an additional entry, the `functions` dictionary. Run time functions are sections of code that can be used to compute specific properties of the flow - in this case the forces and moments on the craft. Run `simpleFoam` on the case and examine the pressure and viscous forces at the different timesteps.

It is quite valuable to examine the residuals from the solution; particularly when running `simpleFoam` where we wish to monitor solution convergence. OpenFOAM writes these to the console, and it is sensible to keep an eye on these to ensure all is well with the simulation. However it is also useful to be able to plot the residuals, and to do this we need to store and process the output. The commands

```
simpleFoam >& log &
tail -f log
```

writes the output from the `simpleFoam` case to a file `log`; `tail -f` writes the content of `log` to the console, thus ensuring that we can see what is happening in real time. Having generated this log file we can then process it with the utility `foamLog` :

```
foamLog log
```

This generates a large number of files specifying how initial and final residuals of the various variables change with time.

[**Q.II.1**] This tutorial is not assessed, but you might like to try altering some of the settings to identify what they do. There are various ways of improving the mesh, including changing the surface max refinement level, including refinement boxes, and changing the boundary layer mesh. Try altering some of these settings and seeing what the differences are.
This exercise can generate a very large computational model which can take quite a while to run. Try using the EPIC system to run the resulting model (http://epic.zenotech.com)