

# Fully turbulent flow around a sphere using OpenFOAM

In this tutorial you will simulate a fully turbulent flow with a Reynolds number of 1 million around a sphere with a radius of 1 m using a given CAD geometry. You are already familiar with OpenFOAM through the EEN-E2001 Computational fluid dynamics course, so the tutorial will focus on the features that you might not have explored yet, i.e. generation of an unstructured mesh using snappyHexMesh and application of turbulence models.

Some of the settings have been taken from Hillier et al (2012), which can be consulted for further details.

## Setting up OpenFOAM

1. Execute the OpenFOAM settings file by typing

```
./opt/openfoam7/etc/bashrc (note that there is a space after the dot)
```

This will set up the command aliases, paths and other environment variables correctly. Note that you must be running bash shell, if you want an error free setup.

2. Check whether you have the default run directory under your home tree by typing

```
ls $FOAM_RUN
```

If you do not get an error, you can skip the next step.

3. If you do not have the default run directory, issue the following command.

```
mkdir -p $FOAM_RUN
```

4. Change to the run directory.

```
cd $FOAM_RUN
```

## Setting up the case based on a template

1. Copy the PitzDaily tutorial as a starting point for the case.

```
cp -r /opt/openfoam7/tutorials/incompressible/simpleFoam/pitzDaily .
```

2. Rename the case directory.

```
mv pitzDaily sphere
```

3. Change to the directory.

```
cd sphere
```

4. Download the STL-file for the geometry from the Tutorial page on MyCourses.

5. Make a directory for the geometry definition.

```
mkdir constant/triSurface
```

6. Copy the STL-file to the directory *triSurface* directory.

## Cleaning the geometry definition

In order to be able to use the STL-file for grid generation, the surface should be watertight. This is often not the case and e.g. the file provided on the MyCourses pages is a typical example of an STL-file with quality problems. Let's clean up the file first.

1. Move to the *triSurface* directory.
2. Rename the STL-file into *sphere\_orig.stl*.
3. Check the quality of the original file.

```
surfaceCheck sphere_orig.stl
```

You will notice that the check complains about various quality issues and dumps the info into various files. You could use this info to clean the file in a CAD software. However, in this case the problems are not severe and we can clean the file using tools provided with OpenFOAM.

4. Convert the STL-file into a new STL-file.

```
surfaceMeshConvert sphere_orig.stl sphere.stl
```

5. Check the quality of the converted file.

```
surfaceCheck sphere.stl
```

You will notice that the converted file does not have any problems.

## Setting up the mesh

We will use *snappyHexMesh* to generate an unstructured hexahedral mesh for the geometry. The process of generating a mesh using *snappyHexMesh* consists of two main steps (more steps might be included depending on the complexity of the case): 1. generation of the background mesh using *blockMesh*, 2. local refinement of the background mesh, removal of the part of the refined background mesh that falls outside of the fluid domain, snapping of the resulting mesh to the geometry definition and addition of a prismatic boundary layer grid. The resolution of the final grid is therefore controlled in two steps: by the resolution of the background mesh and by the extent and level of refinements.

In this case we will generate a very simple cartesian brick type background mesh first. We will cover only one quarter of the sphere. We will refine the mesh close to the sphere surface by specifying a set level of refinements on the sphere surface and we will additionally refine the mesh in the wake of the sphere by specifying a refinement box with a set level of refinements inside the box. We will then add layers of prismatic elements for the boundary layer.

# Generating the background mesh

1. Move to the *system* directory.
2. Replace the *blockMeshDict* file with the following.

```
/*-----* C++ *-----*\
|=====|
|  \ \ /  F i e l d      | OpenFOAM: The Open Source CFD Toolbox
|  \ \ /  O p e r a t i o n | Version: 4.0
|  \ \ /  A n d           | Web:      www.OpenFOAM.org
|  \ \ /  M a n i p u l a t i o n |
\*-----*\
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    object       blockMeshDict;
}
// * * * * * //

convertToMeters 1;

vertices
(
(-5 0 0 )
(-5 0 20)
(-5 20 20)
(-5 20 0 )
(20 0 0 )
(20 0 20)
(20 20 20)
(20 20 0 )
);

blocks
(
hex (0 4 7 3 1 5 6 2) (13 12 12) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
inletWall
{
    type patch;
    faces
    (
        (0 1 2 3)
        (5 6 2 1)
        (7 3 2 6)
    );
}
sym
{
    type symmetry;
    faces
    (
        (7 4 0 3)
    );
}
);
```

```

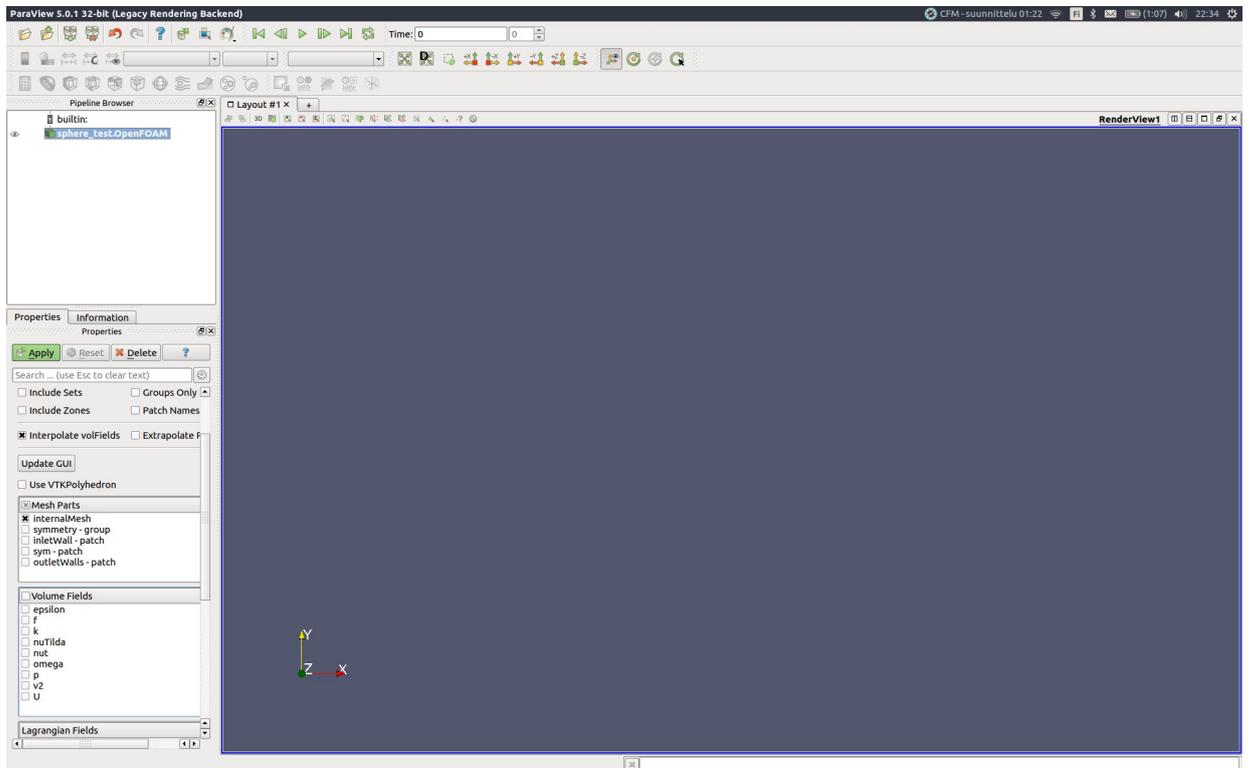
        (4 5 1 0)
    );
}
outletWalls
{
    type patch;
    faces
    (
        (7 6 5 4)
    );
}
);

mergePatchPairs
(
);

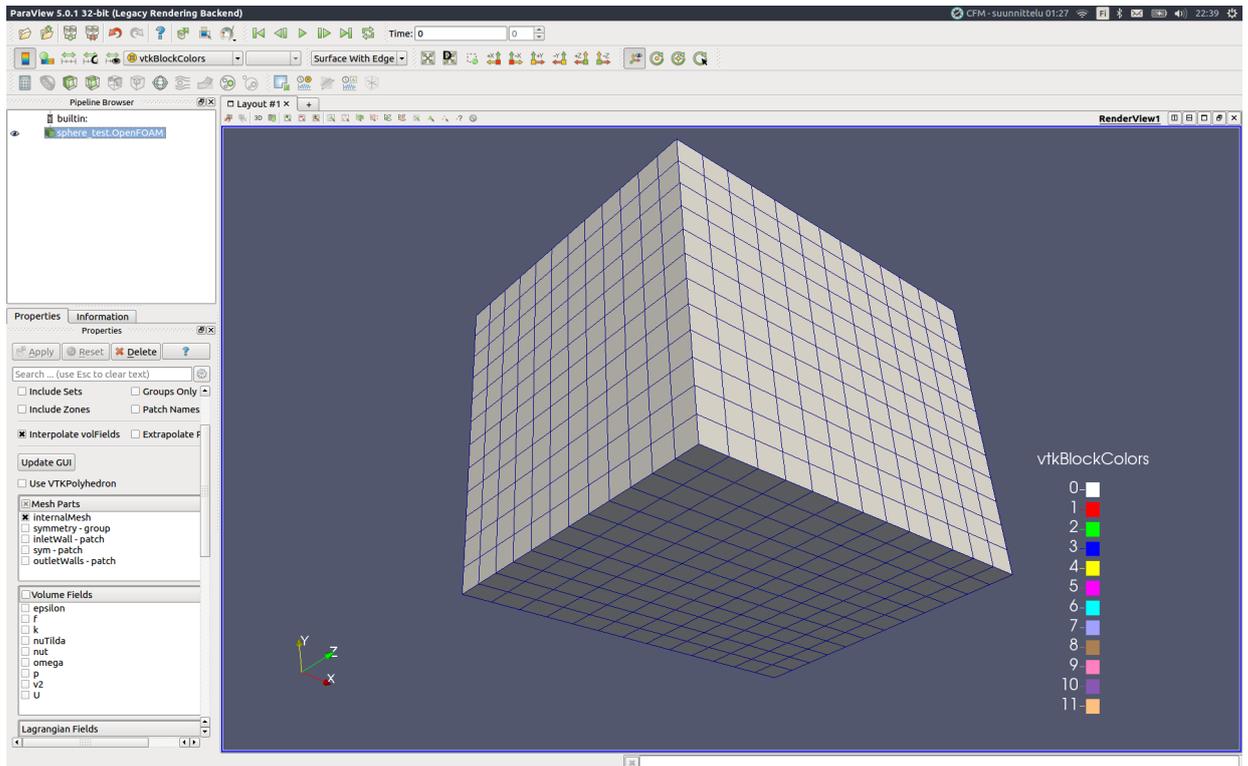
// ***** //

```

3. Generate the background mesh by **moving into the main case directory** and by issuing *blockMesh*
4. Check the background mesh in ParaView by running *paraFoam*
5. In ParaView uncheck all the volume fields by clicking on the checkmark next to the *Volume Fields* until none of the fields are selected.



6. Click *Apply*.
7. Select *Surface with edges* as the drawing style in the drop-down menu in the top toolbar. The background mesh should look like this.



8. Close Paraview.

## Generating the final mesh

1. Copy the *snappyHexMeshDict* example file from the motorbike tutorial into the *system* directory

```
cp /opt/openfoam7/tutorials/mesh/snappyHexMesh/motorBike/system/snappyHexMeshDict
system/
```

2. Modify the *snappyHexMeshDict* based on the following setup.
  - a) Set up a correct geometry file and modify the refinement box by changing the *geometry* section of the file. We should use the cleaned STL-file. However, the dimensions of the file are millimeters and the radius of the sphere in the file is 100 mm. We do not need to modify the STL-file, but instead we can tell snappyHexMesh that we want to scale the geometry data.

In the longitudinal direction the refinement box will start at the centre of the sphere and extend 5 diameters from the centre into the wake of the sphere. In transversal direction the refinement will extend to 1.5 diameters from the centre line and the box will reach slightly out of the domain to avoid any problems at the domain interfaces.

The *geometry* section should look like this.

```
geometry
{
  sphere.stl
  {
    type triSurfaceMesh;
    scale 0.01;
    name sphere;
```

```

    }
    refinementBox
    {
        type searchableBox;
        min (0 -0.5 -0.5);
        max (10 1.5 1.5);
    }
};

```

- b) Change the number of cells between different levels to make the change between levels more gradual.

```
NCellsBetweenLevels 4;
```

- c) Clear the *features* section, which specifies explicit feature edge refinement, because the geometry in this case is smooth and does not contain any sharp edges. After the modification, it should look like this.

```

features
(
);

```

- d) Change the name of the surface, on which the mesh should be refined and increase the level of refinements. We want to apply seven levels of refinement everywhere on the sphere surface, so we set the same values for the minimum and maximum levels of refinement. These are controlled in the *refinementSurfaces* section, which should look like this.

```

refinementSurfaces
{
    sphere
    {
        // Surface-wise min and max refinement level
        level (7 7);
    }
}

```

- e) Under the *refinementRegions* section increase the number of refinement levels.

```
levels ((1E15 6));
```

- f) Make sure that the *locationInMesh* specifies a point that is inside the fluid domain. You can e.g. specify it as

```
locationInMesh (-2.9 0.112 0.045);
```

- g) Specify the following settings for the snapping of the mesh to the sphere surface. Note that we comment out the options for feature snapping, as these are not used.

```

snapControls
{
    //- Number of patch smoothing iterations before finding correspondence
    // to surface
    nSmoothPatch 5;

    //- Relative distance for points to be attracted by surface feature point
    // or edge. True distance is this factor times local
    // maximum edge length.

```

```

tolerance 4.0;

//- Number of mesh displacement relaxation iterations.
nSolveIter 0;

//- Maximum number of snapping relaxation iterations. Should stop
// before upon reaching a correct mesh.
nRelaxIter 5;

// Feature snapping

    //- Number of feature edge snapping iterations.
    // Leave out altogether to disable.
    //nFeatureSnapIter 10;

    //- Detect (geometric only) features by sampling the surface
    // (default=false).
    //implicitFeatureSnap false;

    //- Use castellatedMeshControls::features (default = true)
    //explicitFeatureSnap true;

    //- Detect points on multiple surfaces (only for explicitFeatureSnap)
    //multiRegionFeatureSnap false;
}

```

h) Change the sizes in the *addLayersControl* section to be absolute sizes by setting `relativeSizes false`;

i) Specify that ten prismatic layers should be generated on the sphere surface for the boundary layer by modifying the *layers* section.

```

layers
{
    "sphere.*"
    {
        nSurfaceLayers 10;
    }
}

```

j) Make the following additional changes to the *addLayersControl* section.

```

expansionRatio 1.2;
finalLayerThickness 0.01;
minThickness 0.0001;
featureAngle 30;
nLayerIter 5000;

```

3. Copy the default settings for mesh quality into the *system* directory.

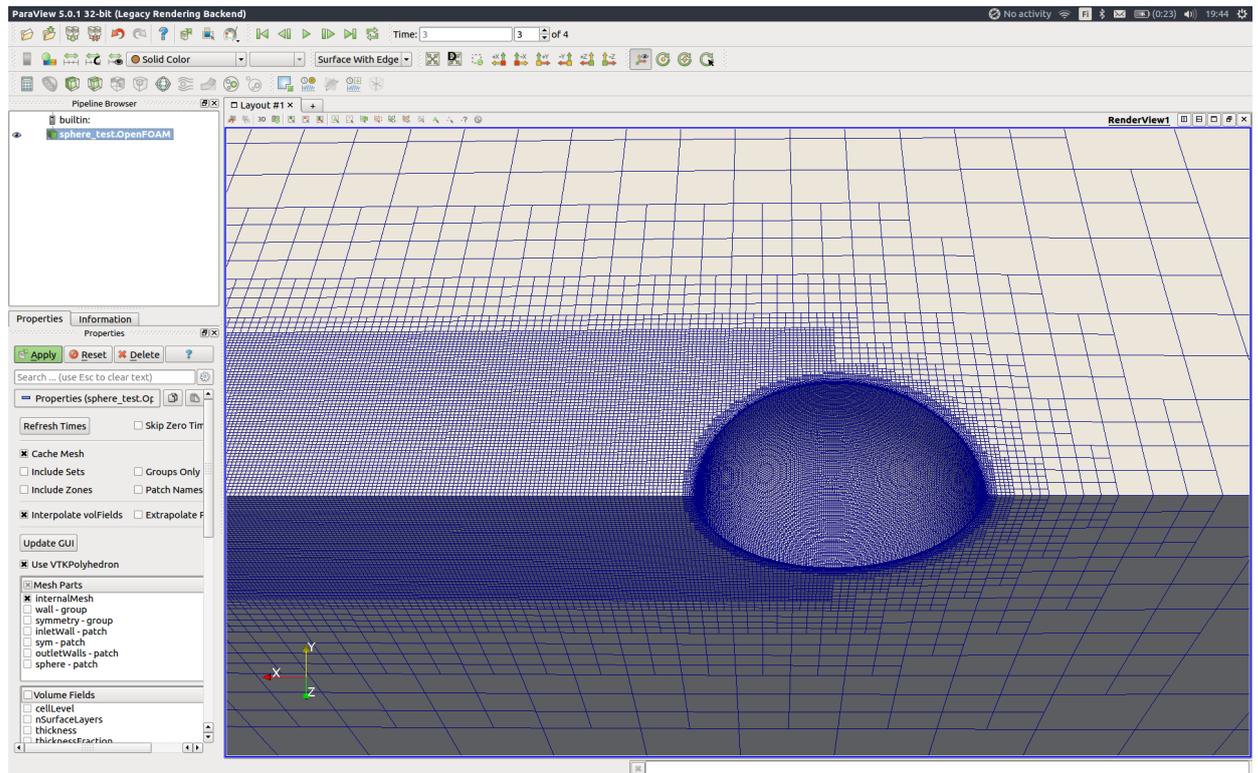
```
cp /opt/openfoam7/etc/caseDicts/mesh/generation/meshQualityDict system/
```

4. Run *snappyHexMesh* in the main case directory.

```
snappyHexMesh
```

5. Check the resulting mesh using ParaView just as you did for the *blockMesh* generated mesh above. Activate the “Use VTK Polyhedron” option while opening the case. Otherwise ParaView will decompose the cells at the refinement boundaries into tetrahedra. The mesh for time 1 is the refined mesh before snapping with the cells lying outside of the fluid

domain removed. The mesh for time 2 is the mesh after snapping to the geometry, but before the addition of prismatic layers for the boundary layer. The mesh for time 3 is the final mesh. The mesh should look like this.



6. Copy the final mesh from the time directory 3 into the *constant* directory by issuing the following in the main case directory.

```
cp -r 3/polyMesh constant/
```

7. Remove the directories containing the intermediate meshes. The directories with intermediate meshes would cause problems in the postprocessing stage, as they only contain the mesh, but not any flow field data.

```
rm -rf 1 2 3
```

## Setting up the models, boundary conditions

We will next specify the fluid properties, the turbulence model and the boundary conditions for the relevant quantities. For turbulence modelling we will use the  $k-\omega$  SST model. In order to specify the correct Reynolds number we can play around with the velocity and the kinematic viscosity of the fluid. In this case we set the flow velocity as 7.5 m/s. The inlet faces will have a fixed value for velocity and the turbulent quantities and the zero gradient condition for pressure (note that in the *blockMeshDict* we specified the external walls parallel to the flow to be in the same group as the inlet face). The outlet face will have the zero gradient condition for the velocity and the turbulent quantities and a fixed value condition for the pressure. Symmetry is enforced on the faces that cut through the sphere. On the sphere surface we will use a fixed value for the velocity, the zero gradient condition for the pressure and wall functions for the turbulent quantities. For the

discretisation schemes we will use the same set as in the `pitzDaily` tutorial, so we do not need to make any modifications in to the `system/fvSchemes` file.

1. Change the kinematic viscosity of the fluid in `constant/transportProperties`.

```
nu [0 2 -1 0 0 0 0] 1.5e-05;
```

2. Change the turbulence model in `constant/turbulenceProperties`.

```
RASModel kOmegaSST;
```

3. Set the initial and boundary conditions for velocity in file `0/U`.

```
internalField uniform (7.5 0 0);

boundaryField
{
    inletWall
    {
        type freestream;
        freestreamValue $internalField;
    }

    outletWalls
    {
        type zeroGradient;
    }

    "sphere*"
    {
        type fixedValue;
        value uniform (0 0 0);
    }

    sym
    {
        type symmetry;
    }
}
```

4. Set the initial and boundary conditions for pressure in `0/p`.

```
internalField uniform 0;

boundaryField
{
    inletWall
    {
        type zeroGradient;
    }

    sym
    {
        type symmetry;
    }

    outletWalls
    {
        type fixedValue;
        value $internalField;
    }
}
```

```

"sphere*"
{
  type      zeroGradient;
}
}

```

5. Set the initial and boundary conditions for turbulent kinetic energy in  $0/k$ .

```

internalField      uniform 0.0021;
boundaryField
{
  inletWall
  {
    type      fixedValue;
    value     $internalField;
  }

  outletWalls
  {
    type      zeroGradient;
  }

  "sphere*"
  {
    type      kqRWallFunction;
    value     $internalField;
  }

  sym
  {
    type      symmetry;
  }
}

```

6. Set the initial and boundary conditions for specific dissipation in  $0/\omega$ .

```

internalField      uniform 0.46;
boundaryField
{
  inletWall
  {
    type      fixedValue;
    value     $internalField;
  }

  outletWalls
  {
    type      zeroGradient;
  }

  "sphere*"
  {
    type      omegaWallFunction;
    value     $internalField;
  }

  sym
  {
    type      symmetry;
  }
}

```

7. Set the initial and boundary conditions for eddy viscosity in *0/nut*.

```
internalField    uniform 0;

boundaryField
{
    inletWall
    {
        type      calculated;
        value     uniform 0;
    }

    outletWalls
    {
        type      calculated;
        value     uniform 0;
    }

    "sphere*"
    {
        type      nutkWallFunction;
        value     uniform 0;
    }

    sym
    {
        type      symmetry;
    }
}
```

8. Remove the unnecessary initial and boundary condition files under the *0* directory.

```
rm -f 0/epsilon 0/f 0/nuTilda 0/v2
```

## Running the simulation

We are now almost ready to run the simulation. We will use the same solver settings as in the *pitzDaily* tutorial for the solution of the linear system of equations, so we can leave *system/fvSolution* pretty much as it is. However, we will change the discretisation scheme for the turbulence model for the beginning of the simulation run to achieve convergence. We will change this back to the ones in the *pitzDaily* tutorial after the initial transient has settled a bit.

1. In the *SIMPLE* section of the *system/fvSolution* file modify the stopping criteria for the *SIMPLE* loop to be more strict.

```
residualControl
{
    p                1e-6;
    U                1e-7;
    "(k|epsilon|omega|f|v2)" 1e-7;
}
```

2. Choose a first order scheme for the discretisation of the convective term of the turbulence equations by modifying the *system/fvSchemes* dictionary.

```
div(phi,k)        bounded Gauss upwind;
div(phi,omega)    bounded Gauss upwind;
```

3. Modify the stopping criteria in *system/controlDict*. If you want to, you can also modify the *writeInterval* to reduce the amount of data being written during the simulation.

```
endTime          100;
```

4. Modify the *functions* section in *system/controlDict* to include the evaluation and logging of forces.

```
functions
{
    #include "forces"
}
```

5. Create a dictionary file *system/forces* which specifies the details of the force evaluation. Templates for this can be found under */opt/openfoam7/etc/caseDicts/postProcessing/forces* directory.

```
forces
{
    type          forces;

    libs ( "libforces.so" );

    writeControl  timeStep;
    timeInterval  1;

    log           yes;

    patches      ( "sphere*" );
    rho          rhoInf;      // Indicates incompressible
    log          true;
    rhoInf       1000;        // Redundant for incompressible

    CofR         (0 0 0);
    pitchAxis    (0 1 0);
}
```

6. Run the simulation.

```
simpleFoam > simpleFoam.log
```

If you want to run the case in parallel using more than one core, instead of the above, do the following.

- a) Copy the template file for the domain decomposition into the *system* directory.

```
cp /opt/openfoam6/etc/caseDicts/preProcessing/decomposeParDict system/
```

- b) Modify the *decomposeParDict* file. You can use a very simple setup as a starting point. Here is an example.

```
numberOfSubdomains 2;
method              scotch;
```

The number of subdomains should correspond to the number of cores you want to use.

- c) Decompose the case for parallel run.

```
decomposePar
```

- d) Run the simulation in parallel.

```
mpirun -np 2 simpleFoam -parallel > simpleFoam.log
```

Here the `-np` option is specifying the number of cores to use.

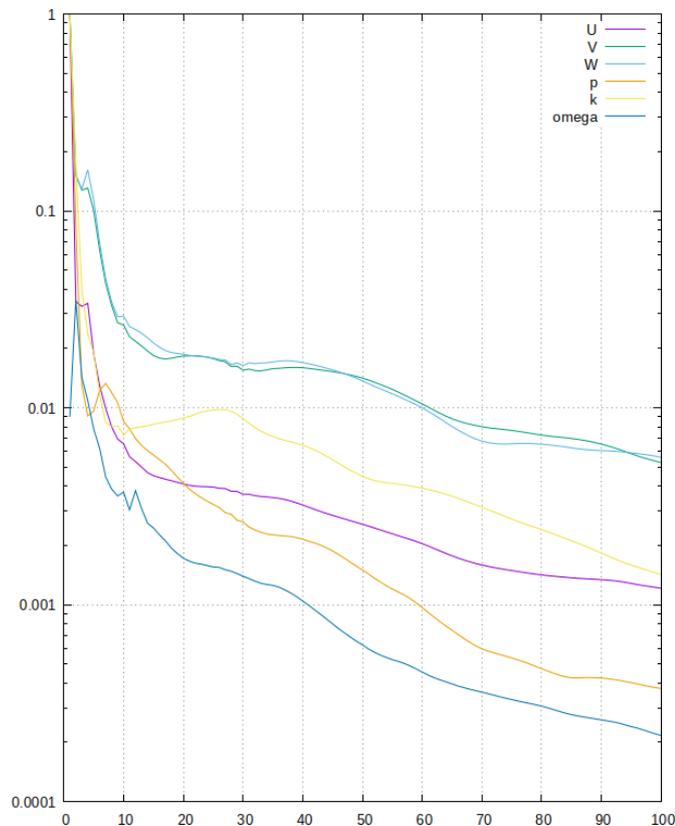
7. Parse the log file in order to get convergence histories as sensible data files.

```
foamLog simpleFoam.log
```

This will create a new directory `logs` which will contain the parsed files.

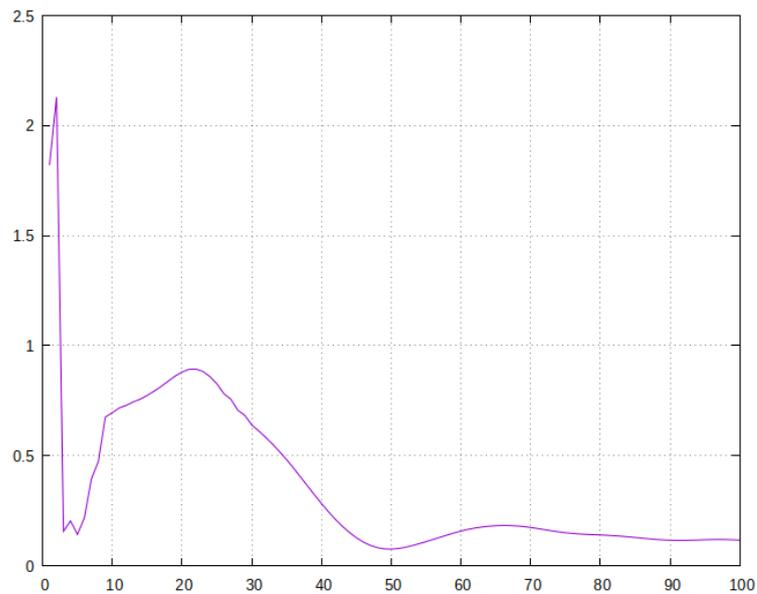
8. Plot the convergence history. In gnuplot you can issue e.g. the following commands.

```
set log y
set grid
p 'logs/Ux_0'      t 'U'      w l,\
  'logs/uy_0'      t 'V'      w l,\
  'logs/Uz_0'      t 'W'      w l,\
  'logs/p_0'       t 'p'      w l,\
  'logs/k_0'       t 'k'      w l,\
  'logs/omega_0'   t 'omega'  w l
```



9. Plot the history of the drag coefficient. In gnuplot you can issue e.g. the following commands.

```
reset
rho = 1000.
V = 7.5
R = 1.
A = 0.25*pi*R**2
CD(x) = x/(0.5*rho*V**2*A)
se grid
p "< sed s/[\\(\\)]//g postProcessing/forces/0/forces.dat"\
  u 1:(CD($2+$5)) not w l
```



## Continuing the simulation

Based on the convergence history and the history of the drag coefficient we can see that the simulation is converging nicely and that the initial transient has reduced significantly. At this point we can try to change the discretisation scheme for the convective term of the turbulent equations back to a more accurate choice and continue the simulation.

1. Change the schemes for the discretisation of the convective term of the turbulence equations by modifying the *system/fvSchemes* dictionary.

```
div(phi,k)          bounded Gauss limitedLinear 1;
div(phi,omega)     bounded Gauss limitedLinear 1;
```

2. Modify the stopping criteria in *system/controlDict*.

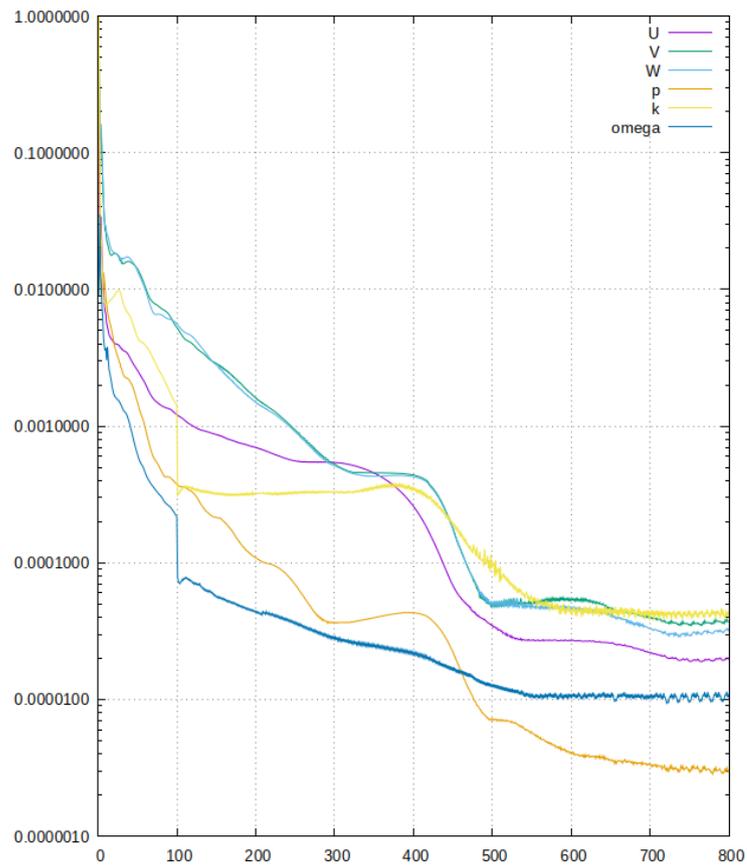
```
startTime 100;
endTime 800;
```

3. Continue the run in either serial or parallel.

```
simpleFoam >> simpleFoam.log
```

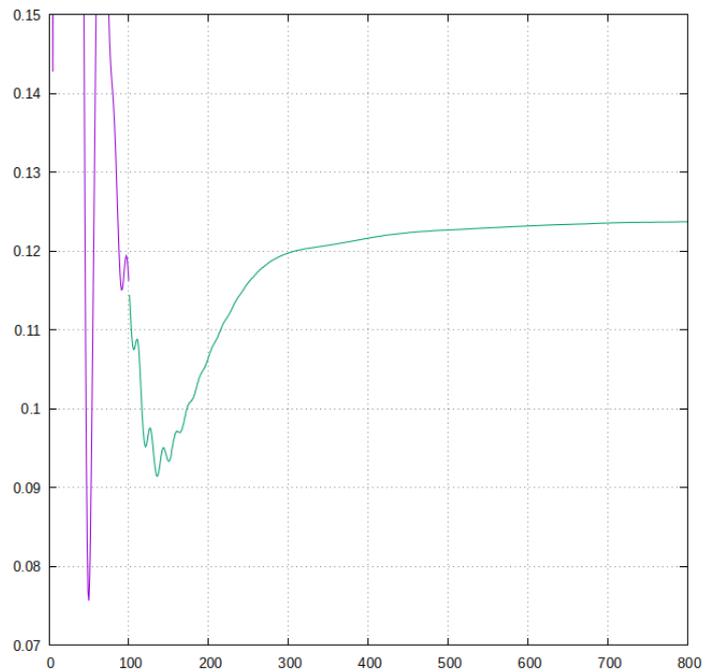
```
mpirun -np 2 simpleFoam -parallel >> simpleFoam.log
```

4. Parse the updated log file according to step 7 in the previous section and plot the convergence history. It should look like this.



5. Plot the evolution of the force.

```
p [][0.07:0.15] "< sed s/[\\(\\)]//g postProcessing/forces/0/forces.dat" u
1:(CD($2+$5)) not w l,"< sed s/[\\(\\)]//g
postProcessing/forces/100/forces.dat" u 1:(CD($2+$5)) not w l
```

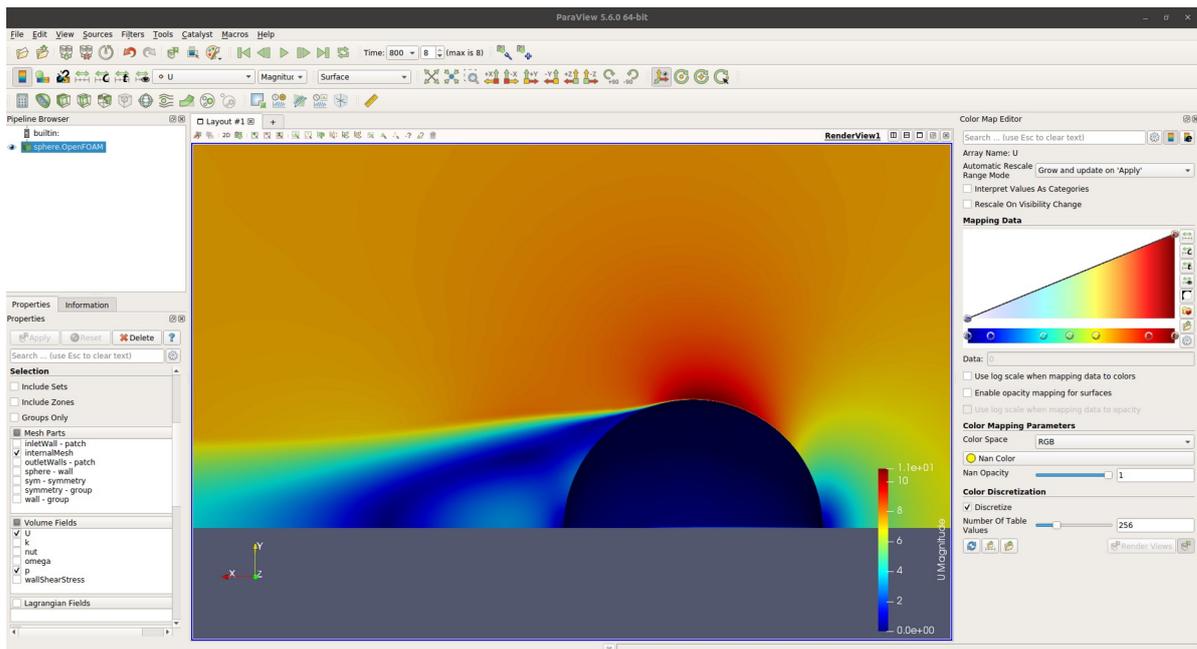


6. If you ran the case in parallel, reconstruct the case to a single directory.

```
reconstructPar
```

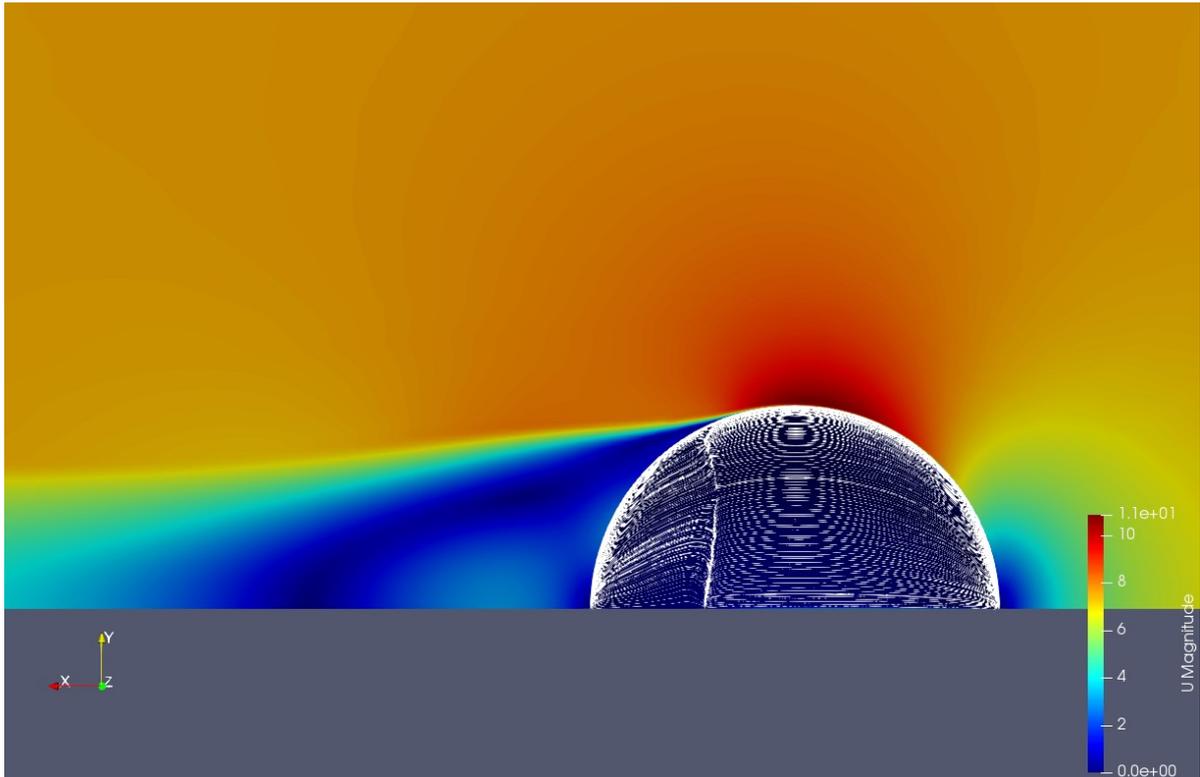
# Analysing the results

1. Calculate the wall shear stress on the sphere surface.  
*simpleFoam -postProcess -func wallShearStress*
2. Open ParaView.
3. Load the case with the internalMesh and at least the  $U$  variable activated.
4. In the toolbar drop-down menu choose to display the magnitude of the velocity. You can change the color scale by clicking the heart stamped folder icon on the right hand edge of the color scale settings. The view should look like this.



5. Let's add limiting streamlines on the sphere surface into the plot. The velocity on the sphere surface is zero, so you cannot produce streamlines on the sphere surface by integrating the velocity vector. However, we have the shear stress vector available and the limiting streamlines are parallel to the shear stress vector. We will use points on the surface as seeding points.
  - a) Add the same case again, but this time activate only the sphere surface and the wallShearStress variable.
  - b) Select the case with just the sphere and generate two transversal slices through the surface. Use the centre as the origin and two offsets under the *Value Range* list: 0 and 0.8.
  - c) Choose *Filters > Search* from the top menu and search for Streamtracer with custom source. Choose the main case part with just the sphere as the input part and the slice part as the seed part (note the radio button on the left hand side of the window).
  - d) In the options for the streamtracer activate the *Surface Streamlines* option and increase the *Maximum Streamline Length* to at least 2. Note that you may need to click the gear next to the search field to show advanced options.

e) Click *Apply*. The result should look like this.



6. In order to plot the pressure and shear stress distributions on the sphere surface load only the sphere surface with pressure and wall shear stress variables activated.
7. Generate additional variables for the angle along the sphere surface, pressure coefficient and shear stress coefficient. Do this by generating successive calculators (button in the toolbar) with the following definitions.

Angle	$\text{acos}(-\text{coordsX})/\text{acos}(-1)*180$
Pressure coefficient	$p/(0.5*7.5^2)$
Shear stress coefficient	$\text{mag}(\text{wallShearStress})/(7.5^2/\text{sqrt}(1e6))$

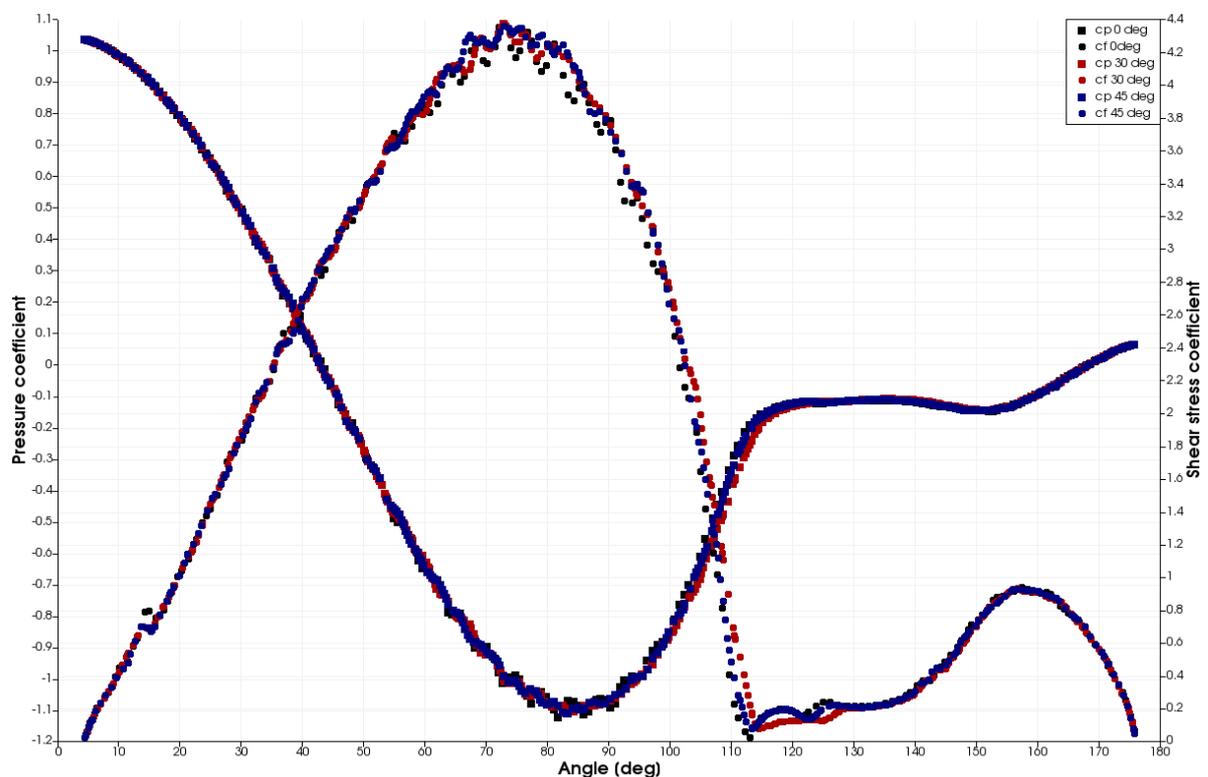
Make sure that you have the previous calculator as an input for the next calculator. This is necessary in order to get all of the new variables propagated through the pipeline.

8. Generate three longitudinal cross-sections through the sphere surface by using the *Slice* operator in the toolbar three times using the last calculator as the input. Use the following specification for the normals and 0, 0, 0 as the origin.

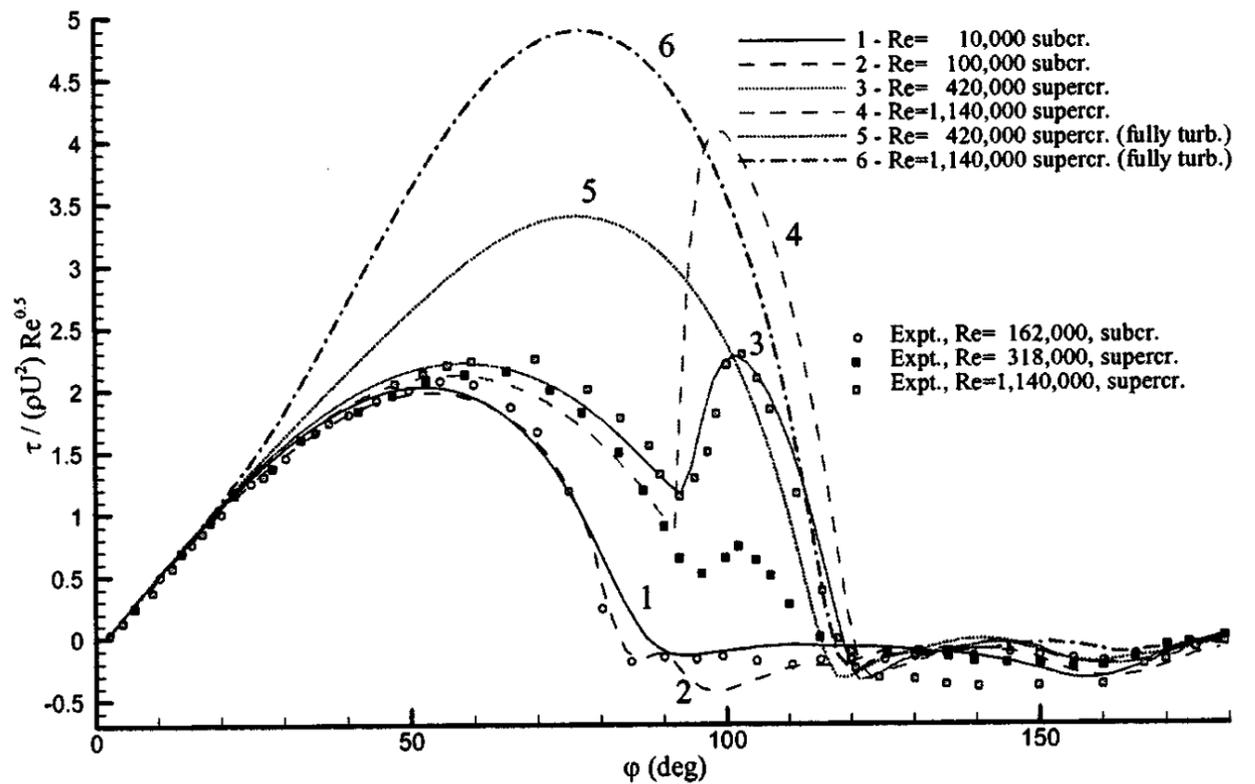
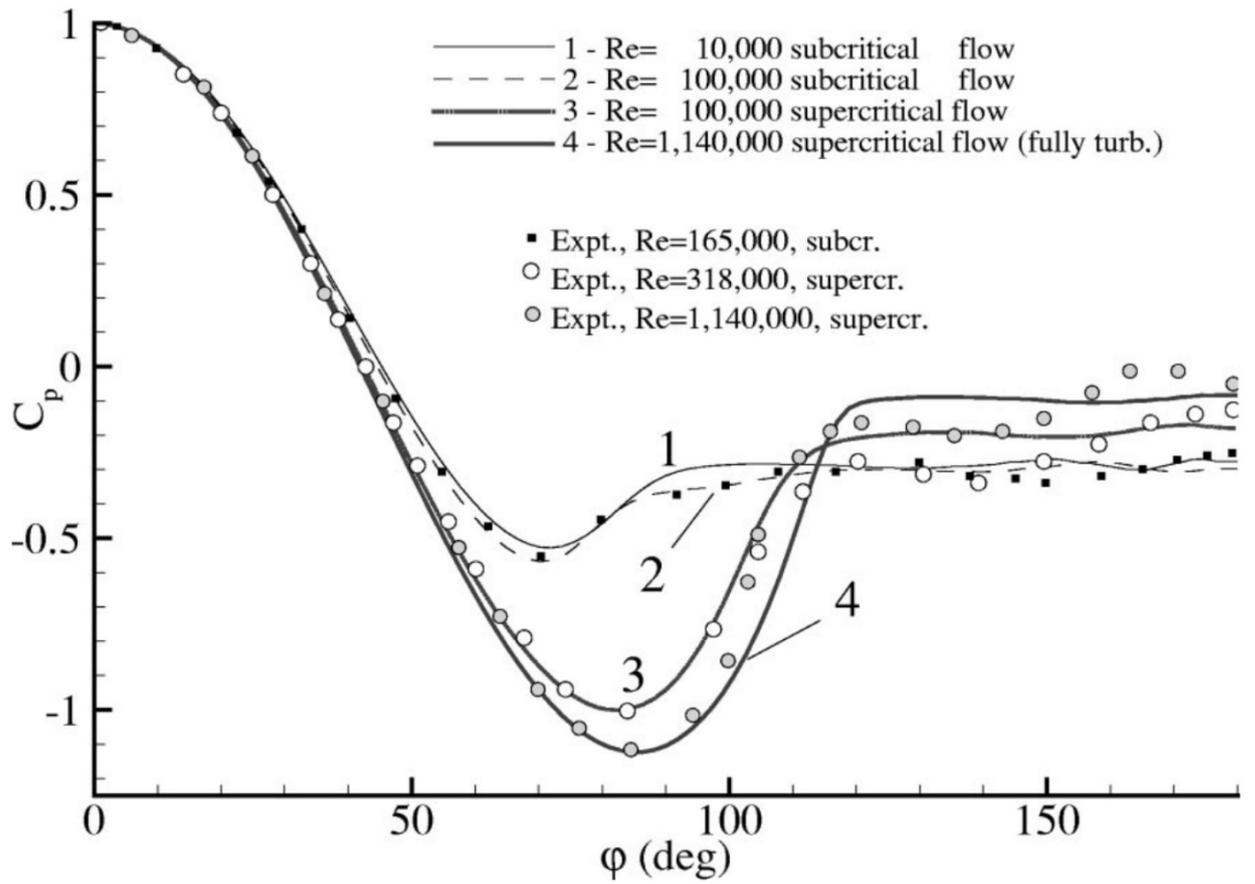
0 degrees	$0, 0, 1$
30 degrees	$0, -1, 1.73205080756888$
45 degrees	$0, -1, 1$

For the 0 degree cross-section use an offset of 1e-10 in the *Value range* field so that the cross-section actually runs through the data.

9. Plot the data by applying the PlotData filter on each of the Slice parts. You can find the filter by choosing Filters > Search and by typing *plot data*. Note that you may need to toggle the visibility of the plotted data in the main pipeline tree.
10. Go through the options for the plot filters and change the *X Array Name* option to your angle variable. Choose the data to plot and the plotting style for each variable using the list of variables and the options under the list. Use bottom-left axis for the pressure coefficient and bottom-right axis for the shear stress coefficient. You can change the name of the variable in the legend by double-clicking the name in the right hand column of the list. You can name the axis under the Left Axis, Right Axis and Bottom Axis headings further down the options view. The end result should be something like this.



The pressure distribution agrees quite nicely with the results presented in Constantinescu and Squires (2004), Numerical investigations of flow over a sphere in the subcritical and supercritical regimes, *Physics of Fluids*, Vol. 16, No. 5 shown below. The skin friction distribution is closer to the numerical fully turbulent curve of the same reference, which is understandable as the simulation does not include transition modelling. Based on the reference results transition seems to occur at around 90 degrees and therefore the simulations are overpredicting the shear stress for the front half of the sphere. Constantinescu and Squires (2004) report that the separation angle for the fully turbulent condition should be  $117 \pm 2$  degrees. The simulation results show that depending on the cross-section, the separation occurs between 112 and 113 degrees. Thus, the separation angle is slightly underestimated. The simulated drag coefficient is around 0.123, which is overestimating the reference value of  $0.102 \pm 0.011$  reported in Constantinescu and Squires (2004). This might be explained by the underestimation of the separation angle and the consequent overestimation of the separation zone.



The study should then be extended with a grid resolution study, study on the influence of the choice of the turbulence model and study on the influence of the domain size and boundary conditions.

## References

Hillier, B., Schram, M. and Maltby, T., (2012), OpenFOAM validation for high Reynolds number flows, Capstone project final report, The University of Melbourne, Department of Mechanical Engineering