CFD WITH OPENSOURCE SOFTWARE

A COURSE AT CHALMERS UNIVERSITY OF TECHNOLOGY
TAUGHT BY HÅKAN NILSSON

Project description:

# A pimpleFoam tutorial for channel flow, with respect to different LES models.

Developed for OpenFOAM-1.6-ext
Requires: FreeCAD, enGrid, pyFoam

*Author:*
OLLE PENTTINEN

*Peer reviewed by:*
EHSAN YASARI
HÅKAN NILSSON

November 4, 2011

# Contents

# Chapter 1

# Introduction

This report will describe how to pre-process, process and post-process a case based on the incompressible Navier-Stokes equations together with approximations based on different LES-models (Large-Eddy Simulation). No validation is performed, since it is just a fictive case. The intention is that the reader should be able to follow the description of the case like a tutorial.

The case is chosen to resemble a generic vortex meter. The principle of the vortex meter is that vorticies are developing downstream a bluff body (in this case represented by a slender rod). The vortex shedding frequency is proportional to the flow rate. The aim of the case description/tutorial is to compare a two different LES models on this simple structure, defined by a squared channel whith unchanged boundary conditions at the inlet. The comparison will be based on samples of the pressure fluctuations downstream a slender rod placed in the center of the channel.

Furthermore the project also aims to give some background to and clarify differences between different turbulence models. This work will be performed on a more theoretical level.

# Chapter 2

# Tutorial

## 2.1 Case definition

The structure on which the simulation will be applied is a very coarse and simplified model of a vortex flowmeter, see Figure 2.1. It is built up of a channel with a cross sectional area of 200x200 mm. The channel length is set to 1000 mm. The slender rod is of equal height as the channel. It is 20x20 mm wide and placed in the center of the channel, with the upstream face positioned 200 mm from the inlet.

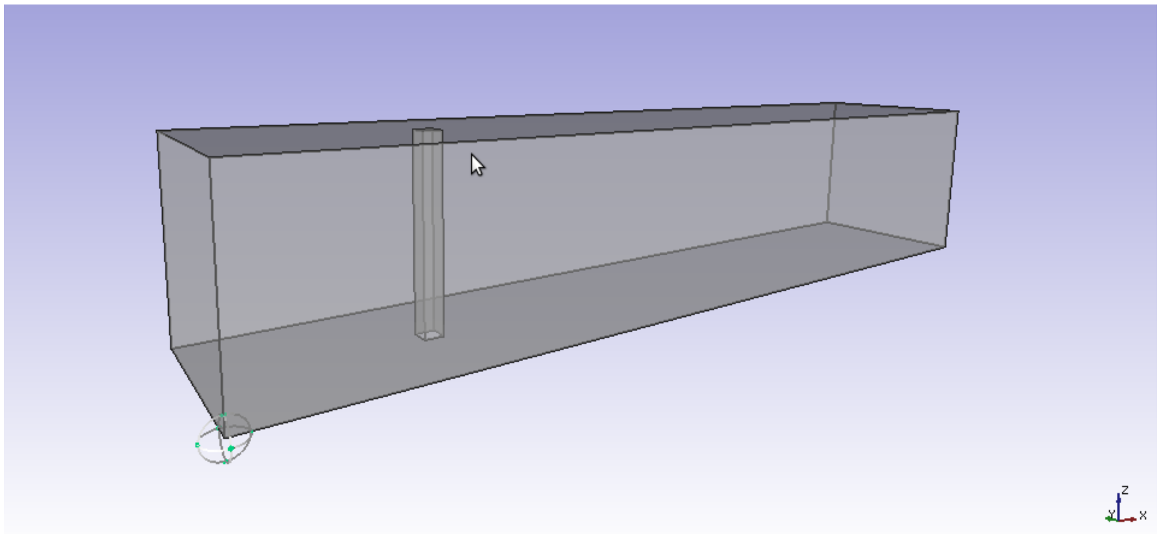

Figure 2.1: Typical CAD drawing which meshing was first intended to be based upon

The fluid of interest is water with a kinematic viscosity, $\nu$, of 1.0e-6 m$^2$/s. The inlet velocity, $v$, is set to 0.25 m/s which would yield a vortex shedding frequency, $F$, of 2.5 Hz according to Equation (2.1). That would yield a simulation time of at least 10 seconds to generate a well developed flow and distinct vorticies. Data need to be generated at around 25 Hz to avoid undersampling.

$$F = \frac{Sr \cdot v}{L} \tag{2.1}$$

where $F$ = Shedding frequency, $Sr$ = Strouhal number = 0.2, $v$ = velocity = 1 m/s, $L$ = bluff body diameter = 0.04 m. The Reynolds number is defined according to Equation (2.2)

$$Re = \frac{vL}{\nu} \tag{2.2}$$

where $L$ equals the hydraulic diameter. For a pipe, that would correspond to the internal diameter. In this case, a channel with a squared cross sectional area is simulated which is why we need to define $L$ as in Equation (2.3)

$$L = \frac{4A}{P} \tag{2.3}$$

where $A$ indicates cross sectional area, and P the wetted perimeter. For the given dimensions, the resulting Reynolds number is evaluated to $Re = 50000$ which is well inside the turbulent region. Hence applying LES to such a problem would be appropriate.

## 2.2 Preprocessing

In the preprocessing step, creating the mesh causes some problems. The work has been divided into two branches, a non working one, and one with better results. Nevertheless, both versions are described below.

### 2.2.1 Meshing - Non working solution

Initially the case structure was intended to be constructed with an open source CAD software. FreeCAD seemed to be a fairly simple alternative. It is available as an Open Source CAD alternative based on OpenCasCade, Qt and Python. There are possibilities to create meshes in STL format, which makes it possible to export the meshes to enGrid (a meshing software with OpenFoam interface). Generating the structure is done by adding two cuboids of appropriate sizes to the drawing. The larger one representing the channel, the smaller representing the rod. The two objects are then joined in a boolean opeation by taking the difference. Meshing suitable for OpenFoam is however difficult in FreeCAD. As mentioned, it is possible to generate a basic mesh, which can be exported as a *.stl file. If the reader is interested in how the structure is created in FreeCAD, a short list of important steps will follow.

1. Download and install FreeCAD which is available from Ubuntu Software Center.

2. Start FreeCAD and create a new empty document. (Ctrl+N)

3. Right click the drawing and select **Navigation Style** > **Inventor Navigation**.
   (It makes the mouse behave as in ParaView for example.)

4. Add two boxes to the drawing.

5. Highlight **Box** in the Project Navigator.

6. Optional: Press **Ctrl+D** and set transparency of **Box** to 50 %.

7. Change the size of **Box** to h=200 mm, l=1000 mm, w = 200 mm.

8. Highlight **Box001** in the Project Navigator.

9. Change the size of **Box001** to h=200 mm, l=20 mm, w = 20 mm.

10. Change the position of **Box001** to x=200 mm, y=90 mm, z=0 mm.

11. Press **Shift** and left click both boxes in the Project Navigator window.

12. Navigate to **Part** > **Boolean** in the menu and select it.

13. Select **Difference** as boolean operation and make sure **Box001** is subtracted from **Box**.

14. Click **Apply**.

15. Click **Meshes** > **Create mesh from geometry...**.

16. Press **Ctrl+E** and export the mesh by naming it with an *.stl extension and selecting **Mesh formats** from the dropdown list before saving.

Having a *.stl file, the open source mesh generator enGrid is chosen as an interface between FreeCAD and OpenFoam. enGrid is a rather new software, developed by Oliver Gloth in 2008. The meshing in enGrid is based on the Netgen libraries, which results in unstructured thetrahedral meshes with possibilites of prismatic boundary layers. The installation of enGrid under Ubuntu 11.04 is not as simple as for FreeCAD. A bit of interaction from the user is needed. To avoid introducing any errors while installing enGrid of the reader is hereby adviced to visit enGits webpage for reliable instructions for different operating systems.

1. Navigate to `http://engits.eu/wiki/index.php/Installation`.

2. Follow the instructions according to your OS.

3. Remark: If enGrid is installed in your home directory, adding

   ```
   alias enGrid='$HOME/engrid/run.bash'
   ```

   to your `.bashrc` file enables you to start the program by just typing enGrid in your terminal.

The aim of using enGrid is to get a proper set of files for OpenFoam. To start with, the earlier created *.stl file is imported to enGrid. When meshing in enGrid, you first need to set boundary codes to the different faces. In this case the structure consists of 10 faces, 8 of them are defined as walls while the other are defined as inlet and outlet patches. One also need to add a volume and point out how the normal vectors of the faces are pointed, i.e on which side of the face the interesting volume is located. That is done by coloring the boundary faces green or yellow (strange). Next, meshing of the boundary faces is performed. If needed, a prismatic boundary layer can be generated afterwords.

When the surface mesh is generated, the next step is to create the volume mesh. Here is where I think enGrid has it weakness. Perhaps it is possible to create a mesh of suitable density but the documentation is rather sparse so it was difficult to figure out how. It seems as if the mesh resolution of the boundary faces completely determines the size of the internal cells and it was difficult to create small enough cells throughout the volume without running into memory problems on an 8 GB machine. A coarse enough mesh distribution to avoid filling up the memory could look something like the one illustrated in Figure 2.2. Unfortunately it is not enough for a reasonable solution to this case.

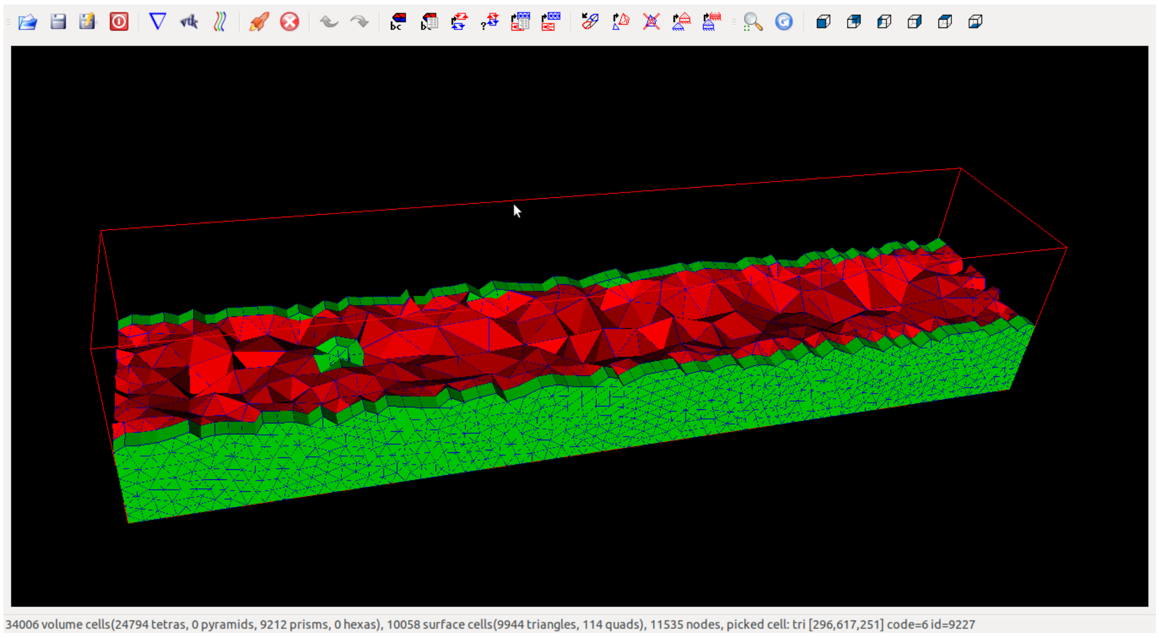
34006 volume cells(24794 tetras, 0 pyramids, 9212 prisms, 0 hexas), 10058 surface cells(9944 triangles, 114 quads), 11535 nodes, picked cell: tri [296,617,251] code=6 id=9227

Figure 2.2: Unable to generate a grid fine enough throughout the volume.

If a solution to the volume meshing is found, the nice thing about enGrid is that it has an interface to OpenFoam. If everything is set up correctly one can export the grid separately, or the grid in combination with solver parameters. It is however a limited number of solvers that are supported, currently simpleFoam and rhoSimpleFoam from version 1.5 to version 1.6.x. Below, enumerated step by step instructions for enGrid is listed, to achieve the mesh in Figure 2.2

1. Start enGrid (by typing enGrid in terminal if you followed the remark)
2. Click **Import** > **STL** and select the *.stl file you previously created
3. Rotate the drawing so you get a decent view of the structure.
4. Hover over the structure with the mouse and select a face by pressing **p**
5. Press **s** to set the boundary code for that face. The Feature Angle can be left at 45.00 but make sure the Boundary Code is incremented for each new boundary.
6. Press **e** to edit boundary codes. For this case it is enough if you have ended up with 10 boundaries.
7. Set the BC-names to ”wall1”-”rod4”, ”rod1”-”rod4” , ”inlet” and ”outlet”
8. Set the BC-types of the wall and rod faces to ”wall” and to ”patch” for the inlet and outlet
9. Press **v** to select which boundaries should be visible.

10. Now it's time to generate the surface mesh. Click on **Mesh** > **edit surface parameters**

11. Mark all faces and insert the following code under "rules for mesh resolution".

    ```
    inlet <OR>
    outlet <OR>
    wall1 <OR>
    wall2 <OR>
    wall3 <OR>
    wall4 <OR>
    rod1 <OR>
    rod2 <OR>
    rod3 <OR>
    rod4 = 2;
    ```

    (Remark: You need to change the number if you want another resolution.)

12. Click **Mesh** > **create/improve surface mesh**

13. Press **e** to edit boundary conditions

14. Add a new volume named "vol"

15. In the new column, select "green", which tells the solver that the control volume is located inside the boundaries.

16. Now, click **Mesh** > **create/improve volume mesh (NETGEN)**

17. Click on "vol" in the dialog and then press **OK**

18. If needed, a prismatic boundary layer can be generated under **Mesh** > **create prismatic boundary layer**

19. The mesh can be examined as in Figure 2.2 by making appropriate settings under **Display options**.

Exporting the mesh to OpenFoam is also possible from enGrid, but since the mesh generated in this tutorial isn't fine enough, that functionality has not been completely investigated. It is also important to point out that the exporting feature is only available for a limited number of solvers. One might have to do some additional changes by hand to make use of the files in an arbritary case.

### 2.2.2 Meshing - Working solution

Since it is difficult to create a mesh with a reasonable cell size ratio in enGrid, it is abandoned and a structured mesh is created manually. The case files can be found in the homepage of the MSc/PhD course in CFD with OpenSource software[1], held at Chalmers in 2011. The `blockMeshDict` file which specifies the location of the verticies, faces and blocks should be placed in `$CASE_DIR/constant/polyMesh/blockMeshDict`. (From here on, the directory of the case will be referred to as `$CASE_DIR`.) Creating a `blockMeshDict` that describes the block distribution shown in Figure 2.3 is enough. Figure 2.3 illustrates a very simple extruded block distribution around a rod (slender dark block in the center). Running blockMesh on such a `blockMeshDict` file results in

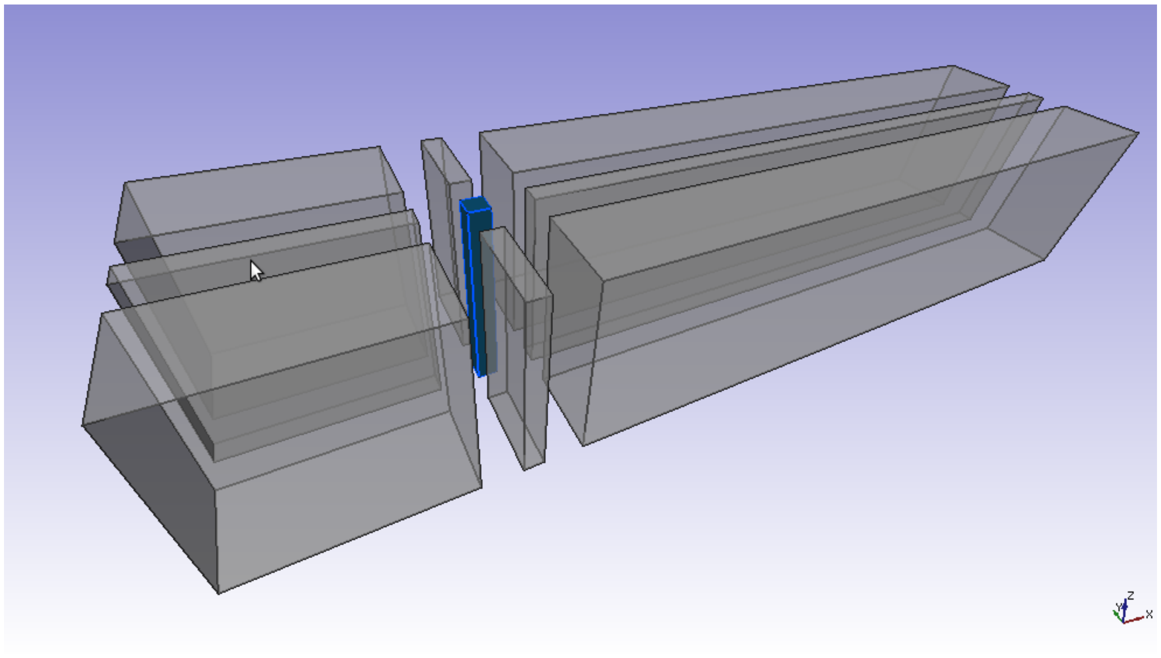

Figure 2.3: Suggested block distribution (not exactly how it was made in this project)

a structured mesh with equal sized cells of 1 cm$^3$ if everything is set up correctly. That is a very coarse grid, which makes the solutions unreliable, but the aim of this project is just to learn the basic procedures in solving a case. However, to resolve the regions around the rod, where generation of verticies was anticipated, a finer mesh is needed. Hence a utility in OpenFoam, called refineMesh is described below.

[1] `http://www.tfd.chalmers.se/~hani/kurser/OS_CFD`

To make use of refineMesh, one first need to define a region on which the refinement shall be performed. This is done in `$CASE_DIR/system/setCellDict`. A typical layout of a `setCellDict` is presented below (When presenting code sections, the file headers are left out for convenience):

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

// Name of set to operate on
name c0;

// One of clear/new/invert/add/delete|subset/list
action new;

// Actions to apply to cellSet. These are all the topoSetSource's ending
// in ..ToCell (see the meshTools library).

topoSetSources
(

    // Cells with cell centre within box
    boxToCell
    {
       box   (0.1 0.05 0) (0.4 0.15 0.2);
    }
);


// ************************************************************************* //
```

The `cellSetDict` is then executed by running `cellSet` in `$CASE_DIR`. This action create files that describes cell sets (or regions), which are located in `$CASE_DIR/constant/polyMesh/sets`

So, having defined the cell sets, it's time to make use of the `refineMesh` utility. `$CASE_DIR/system/refineMeshDict` is defined according to:

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

// Cells to refine; name of cell set
set c0;

// Type of coordinate system:
// - global : coordinate system same for every cell. Usually aligned with
//   x,y,z axis. Specify in globalCoeffs section below.
// - patchLocal : coordinate system different for every cell. Specify in
//   patchLocalCoeffs section below.
coordinateSystem global;
//coordinateSystem patchLocal;


// .. and its coefficients. x,y in this case. (normal direction is calculated
// as tan1^tan2)
globalCoeffs
{
    tan1 (1 0 0);
    tan2 (0 1 0);
tan3 (0 0 1);
}

patchLocalCoeffs
{
    patch outside;  // Normal direction is facenormal of zero'th face of patch
    tan1 (1 0 0);
}

// List of directions to refine
directions
(
    tan1
    tan2
    tan3
//normal
);

// Whether to use hex topology. This will
// - if patchLocal: all cells on selected patch should be hex
// - split all hexes in 2x2x2 through the middle of edges.
useHexTopology  true;

// Cut purely geometric (will cut hexes through vertices) or take topology
// into account. Incompatible with useHexTopology
geometricCut    false;

// Write meshes from intermediate steps
writeMesh       false;

// ************************************************************************* //
```

Here, we see that `refineMeshDict` makes use of the previously defined cell set c0. The utility is executed by typing `refineMesh -dict -overwrite` in `$CASE_DIR`. As seen in the comments of `refineMeshDict`, the utility will divide the cells of the defined region in 2x2x2 parts. These actions will result in a finer grid near the rod, see Figure 2.4. (Screen resolution might affect the appearance.)

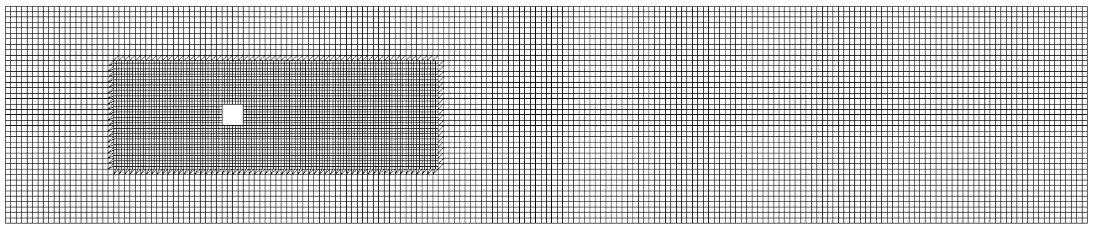

Figure 2.4: Result from refineMesh on mesh specified in the blockMeshDict file

### 2.2.3 Setting up the boundary conditions

The boundary conditions are defined in files located in `$CASE_DIR/0` named `U`, `epsilon`, `k`, `nuSgs`, `nuTilda`, `nut` and `p`. The content of these files will not be presented in detail here.

### 2.2.4 Setting up the solver

To set up information for the solver there is a need to do some changes to the files located in the `$CASE_DIR/system` directory. To start with, `$CASE_DIR/system/controlDict` is specified as follows.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     pimpleFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         15;

deltaT          0.005;

writeControl    adjustableRunTime;

writeInterval   0.04;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression uncompressed;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

adjustTimeStep  yes;

maxCo           1;

functions
{
    probes
    {
        // Where to load it from
        functionObjectLibs ( "libsampling.so" );

        type            probes;

        // Name of the directory for probe data
        name            probes;

        // Write at same frequency as fields
        outputControl   outputTime;
        outputInterval  1;

        // Fields to be probed
        fields
        (
            p U
        );

        probeLocations
        (
            ( 0.01 0.1 0.1 )        // at inlet
            ( 0.19 0.1 0.1 )   // at front face of rod
            ( 0.23 0.1 0.1 )    // at back face of rod
            ( 0.99 0.1 0.1 )        // at outlet
        );
    }
}

// ************************************************************************* //
```

As seen, pimpleFoam[2] is selected as solver, with an adjustable timestep. The timestep is adjusted so the Courant number doesn't exceed 1. Some probes are added for pressure and velocity for post-processing. Probe number three with coordinates `( 0.23 0.1 0.1 )` is located right behind the rod and will later be used to compare the time history of the pressure for two different turbulence models.

Then, a closer look at the `$CASE_DIR/system/turbulenceProperties` is needed.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

simulationType LESModel;

// ************************************************************************* //
```

It is a rather short file, only specifying that a LES model will be used for turbulence modelling. This tells the solver to create a base class of type `LESModel`. The definition of `LESModel` is found in `$FOAM_SRC/turbulenceModels/incompressible/LES/LESModel`. That class "defines the basic interface for an incompressible flow SGS[3] model, and encapsulates data of value to all possible models." according to it's header. If the `simulationType` is set to `LESmodel` a `$CASE_DIR/system/LESProperties` is needed, where the LES model has to be defined. It consists of the following lines.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

LESModel        Smagorinsky;

delta           cubeRootVol;

printCoeffs     on;

cubeRootVolCoeffs
{
    deltaCoeff      1;
}
```

[2]pimpleFoam will be discussed in Chapter 2.3
[3]SGS=Sub Grid Scale

```
PrandtlCoeffs
{
    delta           cubeRootVol;
    cubeRootVolCoeffs
    {
        deltaCoeff      1;
    }

    smoothCoeffs
    {
        delta           cubeRootVol;
        cubeRootVolCoeffs
        {
            deltaCoeff      1;
        }

        maxDeltaRatio   1.1;
    }

    Cdelta          0.158;
}

vanDriestCoeffs
{
    delta           cubeRootVol;
    cubeRootVolCoeffs
    {
        deltaCoeff      1;
    }

    smoothCoeffs
    {
        delta           cubeRootVol;
        cubeRootVolCoeffs
        {
            deltaCoeff      1;
        }

        maxDeltaRatio   1.1;
    }

    Aplus           26;
    Cdelta          0.158;
}

smoothCoeffs
{
    delta           cubeRootVol;
    cubeRootVolCoeffs
    {
        deltaCoeff      1;
    }
    maxDeltaRatio   1.1;
}


// ************************************************************************* //
```

Here, the selection of LES model is made by setting the `LESModel` variable. For this project, two models have been compared, so the variable has been set to `Smagorinsky` in one of the cases and to `oneEqEddy` in the other. These models are defined in the `$FOAM_SRC/turbulenceModels/incompressible/LES/` directory. The models are further explained in Chapter 3

## 2.3 Processing the case with pimpleFoam

Refering to the exact definition[4] of pimpleFoam it is a large time-step transient solver for incompressible flow using the PIMPLE (merged PISO-SIMPLE) algorithm. PISO is an acronym for Pressure Implicit Splitting of Operators for time dependent flows while SIMPLE stands for Semi-Implicit Method for Pressure Linked Equations which is used for steady state problems[5]. In the SIMPLE algorithm a pressure correction term is used while the velocity corrections are neglected because they are unknown. This results in rather slow convergence. The PISO algorithm also neglects the velocity correction in the first step, but then performs one in a later stage, which leads to additional corrections for the pressure[6].

The simulation is run by simply typing `pyFoamPlotRunner.py pimpleFoam >& log` in the `$CASE_DIR`. `pyFoamPlotRunner.py` is part of the `pyFoam` package. If it has been set up correctly, it will generate residual plots while the simulation is performed. These plots will not be presented in this report. Judging from the logfiles, the oneEqEddy model takes longer time than Smagorinsky to execute. Comparing the execution times at a simulation time of 10 s for example, Smagorinsky has run for 7146.36 s while oneEqEddy shows a value of 8740.53 s.

[4]http://www.openfoam.com/features/standard-solvers.php

[5]http://projects.exeter.ac.uk/fluidflow/Courses/ComputationalEng3213-4/slidesCFD/lecture3-slides.pdf

[6]Joel H. Ferziger / Milovan Peric, Computational Methods for Fluid Dynamics, $3^{rd}$ Edition, 2002 Springer

## 2.4 Post-processing

As earlier described in the `controlDict` file, probes have been set up to sample the pressure at well defined positions. It is interesting to look at data gathered with the probe located at position `( 0.23 0.1 0.1 )` which would be located approximately at the position of the pressure sensor of a vortex meter. The sampled signal is plotted vs time. This is done for two LES models (Smagorinsky and oneEqEddy) to show possible differences between the solutions. The pressure samples are located in `$CASE_DIR/probes/0/p`. Using `gnuplot`, Figure 2.5 is created which shows a difference in pressure readings from the two simulations.

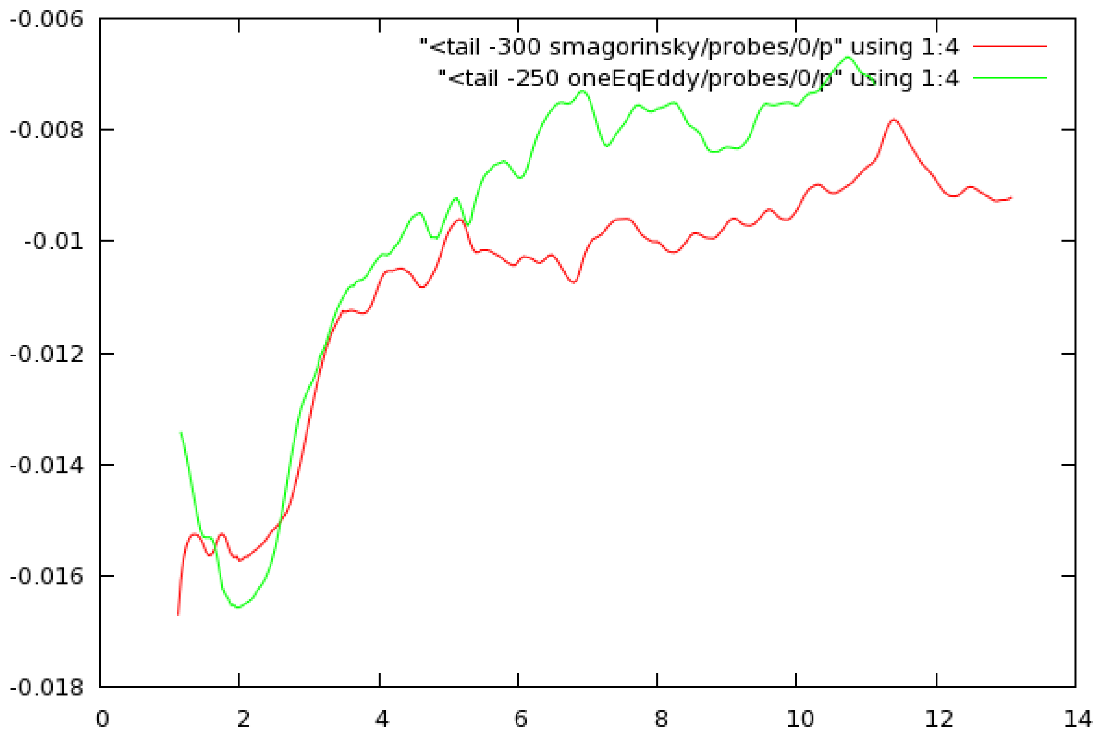

Figure 2.5: Plot of pressure on the back face of the rod for two different LES models

At a later stage, one might be interested in plotting it vs frequency as well, but the dataset is probably to sparse to get any interesting information from these cases.

The pressure distribution can also be illustrated graphically in ParaView. In Figure 2.6 the pressure distrubition after a simulation time of 8 s is illustrated.

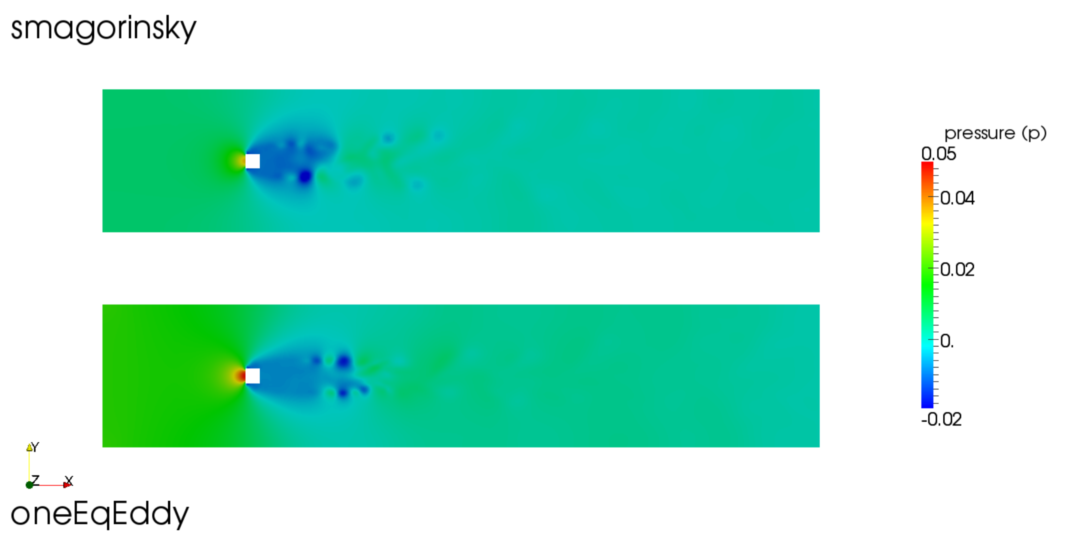

Figure 2.6: Pressure comparison at $t = 8s$, Smagorinsky vs oneEqEddy

The velocity distribution for the same time is illustrated in Figure 2.7. Here the two models clearly show different results.

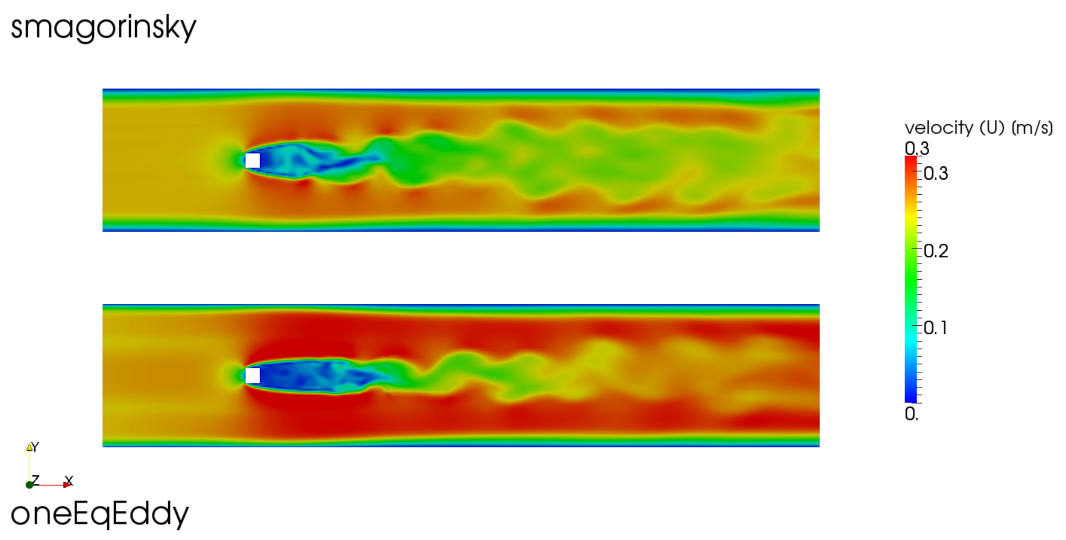

Figure 2.7: Velocity comparison at $t = 8s$, Smagorinsky vs oneEqEddy

## 2.5 Reduction of the near wall eddy viscosity

### 2.5.1 General discussion regarding the Smagorinsky model

According to Ferziger and Peric[7] the Smagorinsky model tends to overestimate the eddy viscosity, $\mu_t$, close to the surfaces of the channel. The eddy viscosity is one component of the effective viscosity, $\mu_{eff}$,which is defined as:

$$\mu_{eff} = \mu + \mu_t \tag{2.4}$$

where $\mu$, the other component, denotes the laminar dynamic viscosity. $\mu_t$ is defined as:

$$\mu_t = C_S^2 \rho \Delta^2 \mid \overline{S} \mid \tag{2.5}$$

where $C_S \simeq 0.2$ is a model parameter and assumed constant. $\rho$ represents the fluid density while $\Delta$ equals the filter cutoff width, i.e. the characteristic length scale of the SGS eddies. $\overline{S}$ represents the strain rate of the resolved field. Ferziger and Peric suggest that the eddy viscosity should be damped by using the model for van Driest damping, commonly used in RANS models:

$$C_S = C_{S0} \left(1 - e^{-n^+/A^+}\right)^2 \tag{2.6}$$

In OpenFoam however, the damping is derived by changing the filter width, depending on the distance from the wall. In `$FOAM_SRC/turbulenceModels/incompressible/LES/vanDriestDelta/vanDriestDelta.C` we see that delta is calculated according to:

$$\Delta = min\left(\Delta_{mesh}, \left(\frac{\kappa}{C_\Delta}\right)\right) y \left(1 - e^{-y^+/A^+}\right) \tag{2.7}$$

where $\Delta_{mesh}$ is the cubic root of the cell volume, $\kappa = 0.4187$ is the von Kármán constant, $C_\Delta = 0.158$, $A^+ = 26$, $y$ represents the distance to the wall, while $y^+$ describes the dimensionless distance to the wall, calculated from the wall shear stress[8]. This also reduces $\mu_t$ near the walls.

### 2.5.2 Applying vanDriest damping to the tutorial case

The eddy viscosity wasn't reduced in the main tutorial of this project since `delta` was set to `cubeRootVol` in `$CASE_DIR/constant/LESProperties` and might be one of the reasons why we see differences between the different LES models. Comparing the two models in Figure 2.7 we see that the fluid elements seem to interact more with the rod, which can be caused by the overestimated eddy viscosity. Interested readers can investigate the effects of a dampened eddy viscosity in the near wall region. If so, change `delta` to `vanDriest` in `$CASE_DIR/constant/LESProperties` instead.

[7]Joel H. Ferziger / Milovan Peric, Computational Methods for Fluid Dynamics, $3^{rd}$ Edition, 2002 Springer
[8]`http://www.modlab.lv/publications/mep2008/pdfs/97-102.pdf`

Making that for a simplified mesh, the pressure fluctuations behind the backface of the rod show a small difference, as seen in Figure 2.8. The differences mainly consists of a slight offset in the signal. The dampened eddy viscosity doesn't seem to have any significant effect on the vortex shedding frequency. It is unclear how much the coarseness of the grid near the wall affects the vortex shedding as well. It could be that the differences in eddy viscosity would have a higher impact if a finer mesh was used. Notice that these pressures are sampled from a different case than

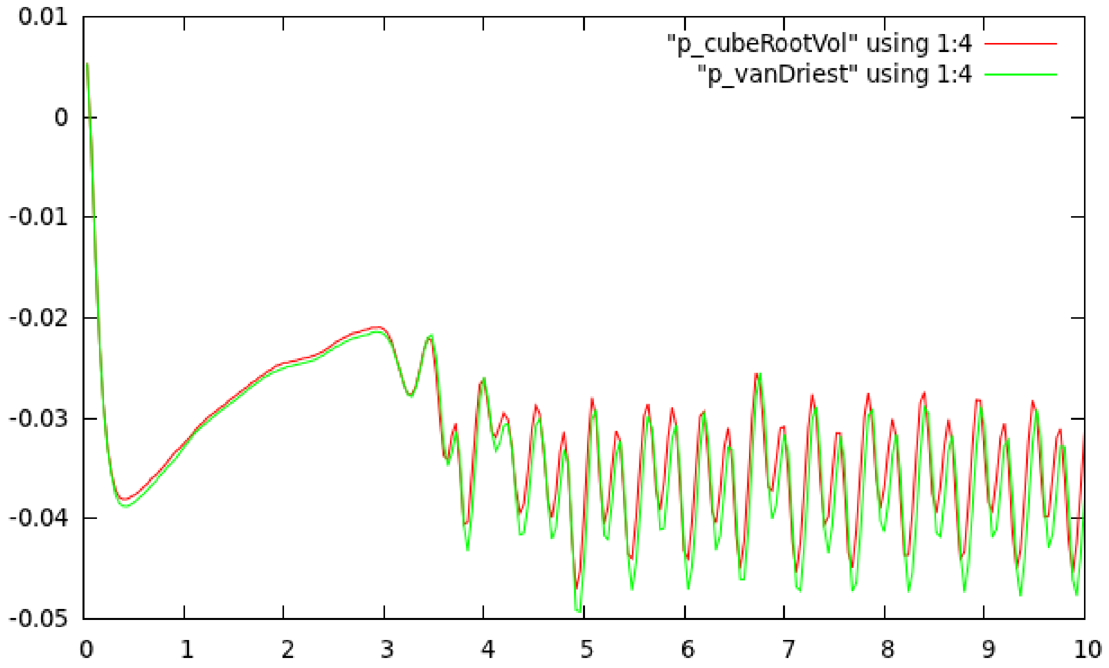

Figure 2.8: Pressure comparison for two different estimations of eddy viscosity.

those presented in Figure 2.6. The mesh used in this case has been flattened and the top and bottom faces are set to empty, which means that the model should not be influenced by any contribution from these boundaries. It might also be a bad idea to use LES on a 2D mesh since it is intended for 3D use. Another alternative than using the vanDriest damping is to implement a dynamic model instead.

# Chapter 3

# Investigation of LES models

Looking at the OpenFoam documentation there are numerous of different LES models to choose from, see Table 3.1. The LES models in OpenFoam are divided into anisochoric and isochoric types, definitions which are not explained in detail in the OpenFoam documentation. Isochoric means that the volume is kept constant, for example in a pressure vessel which doesn't expand though temperature is changing[1]. Anisochoric models would then describe the opposite, i.e. consider a ballon where the volume is changing as soon as pressure or temperature change. The channel in the tutorial could be thought of as a constant volume since it is not expanding. Since the flow is incompressible it is assumed that an isochoric model would be applicable to the case.

| Isochoric LES turbulence models | |
|---|---|
| Smagorinsky | Smagorinsky model |
| Smagorinsky2 | Smagorinsky model with 3-D filter |
| dynSmagorinsky | Dynamic Smagorinsky |
| homogenousDynSmagorinsky | Homogeneous dynamic Smagorinsky model |
| dynLagrangian | Lagrangian two equation eddy-viscosity model |
| scaleSimilarity | Scale similarity model |
| mixedSmagorinsky | Mixed Smagorinsky/scale similarity model |
| dynMixedSmagorinsky | Dynamic mixed Smagorinsky/scale similarity model |
| kOmegaSSTSAS | k-w-SST scale adaptive simulation (SAS) model |
| oneEqEddy | k-equation eddy-viscosity model |
| dynOneEqEddy | Dynamic k-equation eddy-viscosity model |
| locDynOneEqEddy | Localised dynamic k-equation eddy-viscosity model |
| spectEddyVisc | Spectral eddy viscosity model |
| LRDDiffStress | LRR differential stress model |
| DeardorffDiffStress | Deardorff differential stress model |
| SpalartAllmaras | Spalart-Allmaras model |
| SpalartAllmarasDDES | Spalart-Allmaras delayed detached eddy simulation (DDES) model |
| SpalartAllmarasIDDES | Spalart-Allmaras improved DDES (IDDES) model |

Table 3.1: Available isochoric LES models in OpenFoam

Judging from the results of the tutorial, the `Smagorinsky` and `oneEqEddy` models behave differently. Looking at them from a theoretical point of view the differences are revealed.

[1]`http://en.wikipedia.org/wiki/Isochoric_process/`

## 3.1 Smagorinsky

The Smagorinsky model, invented in 1969, is one of the earliest sub grid scale (SGS) models. The SGS represents the spatial region below the resolved length scale. The sparse description of the Smagorinsky model in `$FOAM_SRC/turbulenceModels/compressible/LES/Smagorinsky/Smagorinsky.H` states that:

```
B = 2/3*k*I - 2*nuSgs*dev(D)
Beff = 2/3*k*I - 2*nuEff*dev(D)

where

D = symm(grad(U));
k = (2*ck/ce)*delta^2*||D||^2
nuSgs = ck*sqrt(k)*delta
nuEff = nuSgs + nu
```

According to this description, `B` represents the local SGS stresses, which is important on a SGS level. In Chapter 3.8.2 of An Introduction to Computational Fluid Dynamics[2] it is stated that Smagorinsky assumed the local SGS stresses $R_{ij}$ to be proportional to the local rate of strain of the resolved flow $\overline{S}_{ij} = \frac{1}{2}\left(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i}\right)$. Hence, $R_{ij}$ is defined in Equation (3.1) as:

$$R_{ij} = -2\mu_{SGS}\overline{S}_{ij} + \frac{1}{3}R_{ii}\delta_{ij} = -\mu_{SGS}\left(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i}\right) + \frac{1}{3}R_{ii}\delta_{ij} \tag{3.1}$$

The term $\frac{1}{3}R_{ii}\delta_{ij}$ ensures that the sum of the SGS stresses is equal to the kinetic energy of the SGS eddies. There exists a bit of confusion about the values of the modeling constants `ck` and `ce` according to a thread in `www.cfd-online.com`[3]. That is probably because the Smagorinsky model is implemented differently in OpenFoam compared to definition in literature. If `B` and $R_{ij}$ are compared, we see that they are similar but not identical. We see that the expressions differ by a factor of two. It can possibly be derived from a different definition of the strain rate tensor, $\overline{S}_{ij} = \frac{1}{2}\left(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i}\right)$, between literature and OpenFoam. Further, `Beff` take the ordinary kinematic viscosity into account. Digging into the code of `Smagorinsky.C` a value of 0.094 is found for `ck`. `ce` equals 1.048 according to `$FOAM_SRC/turbulenceModels/compressible/LES/GenEddyVisc/GenEddyVisc.C`. According to literature, Equation (3.2) is an expression for the whole stress tensor which also includes the cross-stresses and Leonard stresses, described in Table 3.2.

$$\tau_{ij} = -2\mu_{SGS}\overline{S}_{ij} + \frac{1}{3}\tau_{ii}\delta_{ij} = -\mu_{SGS}\left(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i}\right) + \frac{1}{3}\tau_{ii}\delta_{ij} \tag{3.2}$$

| Term | Definition | Interpretation |
|---|---|---|
| Leonard stresses | $L_{ij} = \rho\overline{\overline{u}_i\overline{u}_j} - \rho\overline{u}_i\overline{u}_j$ | Effects at resolved scale |
| Cross-stresses | $C_{ij} = \rho\overline{\overline{u}_i u'_j} + \rho\overline{u'_i\overline{u}_j}$ | Interactions between SGS eddies and resolved flow |
| LES Reynolds stresses | $R_{ij} = \rho\overline{u'_i u'_j}$ | Interactions between SGS eddies |

Table 3.2: SGS stress contributions

It is rather unclear if the introduction of the laminar kinetic viscosity in the expression for `Beff` makes it identical to Equation (3.2).

[2]An Introduction to Computational Fluid Dynamics, The Finite Volume Method, 2nd Edition, H. K. Wersteeg, W. Malalasekera, 1995-2007, Pearson Education Limited

[3]`http://www.cfd-online.com/Forums/openfoam-solving/65838-smagorinsky-model-details.html`

## 3.2 oneEqEddy

The oneEqEddy model on the other hand, uses a modeled balance equation for the SGS turbulent energy to simulate the behaviour of k. Looking in the `$FOAM_SRC/turbulenceModels /compressible/LES/oneEqEddy/oneEqEddy.H` file, the following explanation is found:

```
        d/dt(k) + div(U*k) - div(nuEff*grad(k))
        =
        -B*L - ce*k^3/2/delta

    and

        B = 2/3*k*I - 2*nuSgs*dev(D)
        Beff = 2/3*k*I - 2*nuEff*dev(D)

    where

        D = symm(grad(U));
        nuSgs = ck*sqrt(k)*delta
        nuEff = nuSgs + nu
```

The eddy viscosity seems to be modelled similarly as in the Smagorinsky model, but here an additional transport equation for the SGS turbulent kinetic energy, Equation (3.3), is solved as well.

$$\frac{\partial}{\partial t}(k) + \frac{\partial}{\partial x_i}(U_i k) - \frac{\partial}{\partial x_i}\left(\nu_{Eff}\frac{\partial}{\partial x_i}(k)\right) = -BL - \frac{c_e k^{3/2}}{\Delta} \tag{3.3}$$

Here, the first term on the left side describes the change of the turbulent SGS kinetic energy with respect to time. The second term describes convection and the third, diffusion. On the right hand side, two different source terms are listed. The first one $-BL$, is left without further description in the `oneEqEddy.H` file. Eugene de Villiers explains the background of the oneEqEddy model in his PhD thesis from 2006[4]. According to his thesis, the right hand side includes a term which he calls $\tau : \overline{S}$, which "represents the decay of turbulence from the resolved scales to the sub grid scales via the energy cascade". Looking into `$FOAM_SRC/turbulenceModels /compressible/LES/oneEqEddy/oneEqEddy.C` the member function `oneEqEddy::correct(const tmp<volTensorField>& gradU)` reveals a difference between the comments in `oneEqEddy.H` and the implementation in `oneEqEddy.C`.

```
void oneEqEddy::correct(const tmp<volTensorField>& gradU)
{
    GenEddyVisc::correct(gradU);

    tmp<volScalarField> G = 2.0*nuSgs_*magSqr(symm(gradU));

    tmp<fvScalarMatrix> kEqn
    (
       fvm::ddt(k_)
     + fvm::div(phi(), k_)
     - fvm::laplacian(DkEff(), k_)
    ==
       G
     - fvm::Sp(ce_*sqrt(k_)/delta(), k_)
    );
```

[4]http://powerlab.fsb.hr/ped/kturbo/OpenFOAM/docs/EugeneDeVilliersPhD2006.pdf

```
    kEqn().relax();
    kEqn().solve();

    bound(k_, kMin_);

    updateSubGridScaleFields();
}
```

Here, $-BL$ is represented by `G` which is defined in Equation (3.4).

$$G = 2.0 * \nu_{SGS}|\overline{S}|^2 \tag{3.4}$$

This corresponds to the decay term described in Eugenes thesis. The last term on the right hand side $-\frac{c_e k^{3/2}}{\Delta}$, corresponds to turbulent dissipation.

## 3.3 kOmegaSSTSAS

Just to give an example of an alternative implementation of a LES model, kOmegaSSTSAS is also mentioned. It is a model, developed in a European founded project called DESider, between 2004 and 2007. This model is based on the kOmegaSST which mixes two turbulence models, the $k - \omega$ model in the near wall region, where vorticity is expected and the $k - \epsilon$ model further out[5]. It also satisfy the Bradshaw assumption that $\tau_{ij} - \tau_{ii}$ in a boundary layer is proportional to $k$. The difference between the kOmegaSSTSAS and kOmegaSST is that they behave differently in unstable conditions. kOmegaSSTSAS, just as kOmegaSST, behaves like a steady state model for stable conditions. When the conditions become unsteady, $\mu_t$ is decreased which make kOmegaSSTSAS behave more like a LES model[6].

[5] `http://www.cfd-online.com/Forums/openfoam-solving/58056-komega-turbulence-model-wwwopenfoamwikinet.html`

[6] DESider-A European Effor on Hybrid RANS-LES Modelling, W. Haase, M. Braza, A.Revell, 2009 Springer-Verlag Berlin Heidelberg