OpenFOAM tutorial: Large Eddy Simulation of a Tilt-rotor wing with Active Flow Control

Mohammad El-Alti
Division of Fluid Dynamics, Department of Applied Mechanics
Chalmers University of Technology, SE-412 96 Göteborg, Sweden
e-mail: mohammad.el-alti@chalmers.se

April 15, 2008

1 Introduction

In this tutorial, The Large Eddy Simulation solver in OpenFOAM is used to predict the time-dependent flow past a tilt-rotor wing. The case is 3D but as an initial condition for the transient problem, a 2D solution using a laminar solver is calculated and mapped into the 3D problem. There are two LES solvers which suit this case. The oodles and the transientSimpleOodles. The latter will be used here. By finishing this tutorial you will learn the following:

Tutorial Objectives

- Importing a 2D mesh from Fluent into OpenFOAM format
- Extruding a 2D mesh into 3D with specified direction, number of layers and width of each layer.
- Creating boundaries/patches for any feature angle, 90 degrees in this case.
- Defining boundary conditions
- Using the appropriate settings for the LES solver.
- Calculating the lift and drag in each time step during the run.
- Creating a patch of any specific faces in the model.
- Using a time-dependet inlet boundary condition, which is our actuator for the Active Flow Control
- Visualizing the velocity and pressure field.

The solvers used

icoFoam is transient solver for incompressible, laminar flow of Newtonian fluids

oodles is an incompressible LES solver with the PISO iterative algorithm for solving the equations for velocity and pressure.

transientSimpleOodles is the same solver as oodles but with the SIMPLE iterative algorithm.

The utilities used

fluentMeshToFoam is a converter of a fluent 2D mesh in ASCII format to OpenFOAM format

extrudeMesh is a mesh generator by extruding an existing patch.

autoPatch is a patch creator which devides the external faces into patches for user defined feature angle

patchTool is a tool where the user can specify faces to create a new path using the model.

liftDrag is a tool which calculates the lift and drag.

decomposePar is a tool which decomposes the mesh and field in order to run a parallel simulation

2 Prerequisites

This tutorial assumes that the user has completed the basic tutorial cases in the OpenFOAM User Guide. This tutorial is for OpenFOAM ver.1.4.1 and OpenFOAM ver. 1.4.1-dev.

3 Problem Description

The problem details and the geometry is shown in Figure 1. The flow is simulated in a wind tunnel where a wing is placed in the middle. The ambient flow velocity is $U_\infty = 5\ m/s$ and the wing chord is 29 cm.

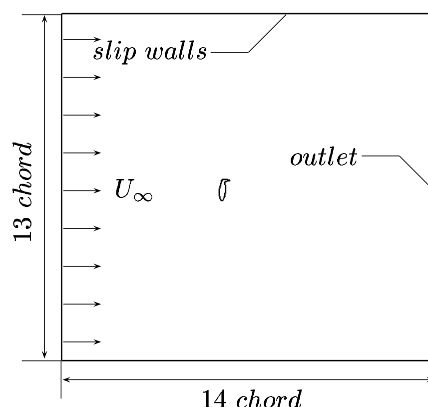

Figure 1: Computational domain.

4 Preparation

Do the following

1. Download the tutorial file LES-AFC.tgz which include the mesh, the transientSimpleOodles solver, the liftDrag utility and other files needed from http://www.tfd.chalmers.se/ hani/kurser/OF_phD_2007/LES-AFC.tgz.

2. put the file into your run directory and type.

```
tar -xzfv LES-AFC.tgz
```

5 Preprocessing

Step 1: Mesh conversion

We are now going to convert the Fluent-ASCII .msh file into OpenFOAM format. There are two utilities for fluent mesh conversion, one for 2D meshes and the other for 3D ones. Enter your run directory and type the following

```
fluentMeshToFoam
```

which will give the following output

```
/*---------------------------------------------------------------------------*\

Usage: fluentMeshToFoam <root> <case> <Fluent mesh file> [-writeSets] [-writeZones] [-scale scale factor]


--> FOAM FATAL ERROR : Wrong number of arguments, expected 3 found 0


FOAM exiting
```

Using this information type the following, we scale the mesh with factor 0.01 because the mesh is in *cm*.

```
fluentMeshToFoam . wing2d wing2d/meshToConvert/mesh2d.msh -scale 0.01
```

This will give you the following output

```
/*---------------------------------------------------------------------------*\

Dimension of grid: 2
Number of points: 33265
Reading points
number of faces: 66271
Reading mixed faces
```

```
Reading mixed faces
Number of cells: 33006
Other readCellGroupData: 2 1 80ee 1 3
Reading uniform cells
Read zone1:2 name:fluid patchTypeID:fluid
Reading zone data
Read zone1:3 name:wall patchTypeID:wall
Reading zone data
Read zone1:5 name:default-interior patchTypeID:interior
Reading zone data

FINISHED LEXING

dimension of grid: 2
Grid is 2-D. Extruding in z-direction by: 10.6759
Creating shapes for 2-D cells
Building patch-less mesh...--> FOAM Warning :
    From function polyMesh::polyMesh(... construct from shapes...)
    in file meshes/polyMesh/polyMeshFromShapeMesh.C at line 577
    Found 66530 undefined faces in mesh; adding to default patch.
done.

Building boundary and internal patches.
Creating patch 0 for zone: 3 start: 1 end: 518 type: wall name: wall
Creating patch 1 for zone: 5 start: 519 end: 66271 type: interior name: default-interior
Creating patch for front and back planes

Adding new patch wall of type wall as patch 0
Patch default-interior is internal to the mesh  and is not being added to the boundary.
Adding new patch frontAndBackPlanes of type empty as patch 1

Default patch type set to empty

Writing mesh... to "constant/polyMesh"  done.

End
```

From the output above we understand that the 2D mesh has been convertet successfully and because all simulations in OpenFOAM should be in 3D the mesh has been extruded one cell in z-direction and frontAndBackPlanes patches has been added. However we want to extrude this mesh from one patch/surface so we have to create all the patches for feature angle 90 using the autoPatch utility. Type

```
autoPath . wing2d 90
```

This will create 7 patches according to feature angle of 90 The new patches with the polymesh is found in the directory $wing2d/5e-05/polyMesh$ so do the following in order to replace the old mesh with old patches with the new ones.

```
cd wing2d
rm -r constant/polyMesh
mv 5e-05/polyMesh constant/
rmdir 5e-05
cd ..
```

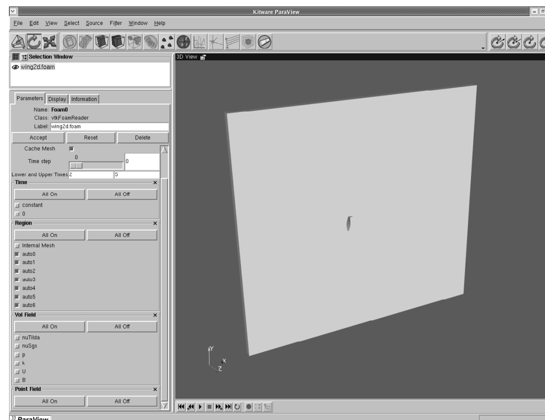

Figure 2: The domain visualized in paraFoam

In order to visualize this we start paraFoam and look at our imported mesh and created patches. Type

```
paraFoam . wing2d
```

Unselect all variables in the Vol Field box and also the internal mesh in the Region box and hit Accept, see Figure 2. To see the mesh edges: Display tab → Display style box → Representation → Wireframe of surface.

We want to extrude the surface closest to origin in z-direction, by hiding and showing the different patches created we find out that *auto6* patch is the plane, closest to origin.

Step 2: Extrusion from a patch to 3D mesh

From step 1 we know that patch *auto6* is the one to extrude from. We will use the *extrudeMesh* utility. By only typing the utility name we get the following information

```
Usage: extrudeMesh <target root> <target case> <nLayers> <overal thickness>
[-mergeFaces] [-sourceCase source case] [-sourcePatch source patch]
[-surface surface file] [-sourceRoot source root]


--> FOAM FATAL ERROR : Wrong number of arguments, expected 4 found 0
```

We need to specify some arguments here. Following the order specified above we enter the following

```
extrudeMesh . wing3d 10 0.04 -sourceCase wing2d -sourcePatch auto6 -sourceRoot .
```

We have specified 10 layers with overall thickness of 4 *cm* and we don't want to merge the original patch with the one created on the other side. So the output will be

```
Extruding layers:
    number of layers 10
    overall thickness 0.04

Create time

Extruding patch "auto6" on mesh "." "wing2d"

Create polyMesh for time = 0

Writing patch as surfaceMesh to "auto6.sMesh"

Mesh bounding box:(-1.6764 -1.6764 -0.0533793) (2.2352 1.9558 -0.0133793)
        with span:(3.9116 3.6322 0.04)
Merge distance   :4e-06

Collapsing edges < 4e-06 ...

End
```

The final mesh is presented in figure 3. Finally we create the patches using *autoPath* utility as described above in order to set the boundary conditions. Type in your run directory

```
autoPatch . wing3d 90
cd wing3d
rm -r constant/polyMesh
mv 5e-05/polyMesh constant/
rmdir 5e-05
cd ..
```

In figure 3 we observe the patches created and the number of cell layers in z-direction.

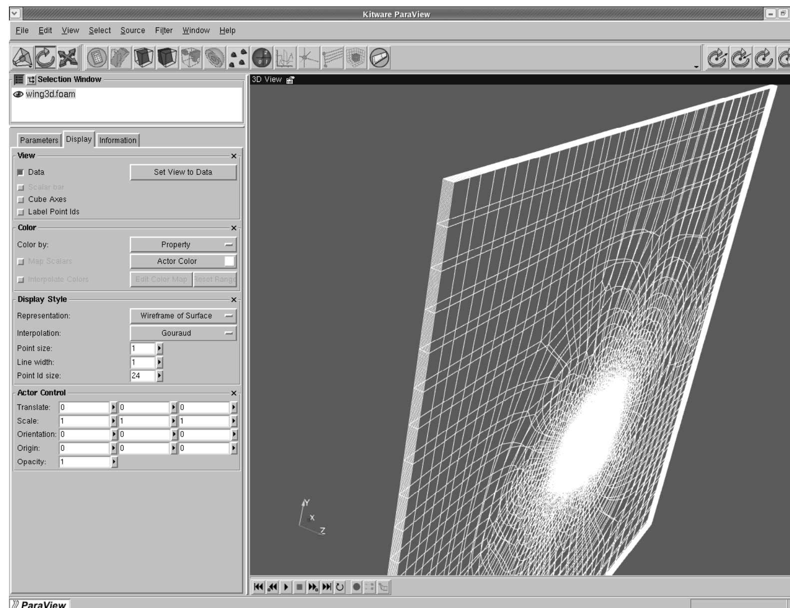

Figure 3: The final created mesh visualized in paraFoam

6 Running

Step 3: Solver set-up

We are now going througth all the settings in the controlDict, fvSchemes and fvSolution dictionaries. Starting with the controlDict shown below, we start from latestTime running with time step $5e^{-5}$ s and stops the simulation at 0.5 s. The data will be written every 0.1 s in compressed and binary format. It is also possible to change these settings during the simulation.

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

application     transientSimpleOodles;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         0.5;

deltaT          5e-05;

writeControl    runTime;

writeInterval   0.01;

purgeWrite      0;
```

```
writeFormat     binary;

writePrecision  6;

writeCompression compressed;

timeFormat      general;

timePrecision   6;

graphFormat     raw;

runTimeModifiable yes;

functions
(
);

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

In the fvSchemes dictionary we try to have second order schemes in order to have accurate solution. As time scheme is defined in the ddtSchemes subdictionary, we chose Crank Nicholson with off-center coefficient of 1 in order to have pure Crank Nicholson. This is a second order scheme. For the gradient and divergence terms we choose Gauss and as interpolation scheme, linear, which is a second order unbounded scheme. For the laplacian terms we choose also the Gauss integration with linear interpolation and corrected surface normal gradient scheme. The dictionary is shown below.

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

ddtSchemes
{
    default         CrankNicholson 1;
}

gradSchemes
{
    default         Gauss linear;
    grad(p)         Gauss linear;
    grad(U)         Gauss linear;
}

divSchemes
{
    default         Gauss linear;
    div(phi,U)      Gauss linear;
    div(phi,k)      Gauss linear;
    div(phi,B)      Gauss linear;
    div(B)          Gauss linear;
    div(phi,nuTilda) Gauss linear;
    div((nuEff*dev(grad(U).T()))) Gauss linear;
}

laplacianSchemes
{
    default         none;
    laplacian(nuEff,U) Gauss linear corrected;
```

```
    laplacian((1|A(U)),p) Gauss linear corrected;
    laplacian(DkEff,k) Gauss linear corrected;
    laplacian(DBEff,B) Gauss linear corrected;
    laplacian(DnuTildaEff,nuTilda) Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
    interpolate(U)  linear;
}

snGradSchemes
{
    default         corrected;
}

fluxRequired
{
    default         no;
    p;
}


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

The linear solvers and the algorithm is controlled in the fvSolution dictionary where we chose Preconditioned conjugate gradient for the pressure equation and the Preconditioned (bi-)conjugate gradient for all other equations. The difference between these linear solvers is that the PCG is for symmetric matrices and PBiCG is for asymmetrix ones.

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

solvers
{
    p PCG
    {
        preconditioner   DIC;
        tolerance        1e-06;
        relTol           0;
    };
    pFinal PCG
    {
        preconditioner   DIC;
        tolerance        1e-06;
        relTol           0;
    };
    U PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-05;
        relTol           0;
    };
    k PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-05;
        relTol           0;
```

```
    };
    B PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-05;
        relTol           0;
    };
    nuTilda PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-05;
        relTol           0;
    };
}

PISO
{
    momentumPredictor yes;
    nCorrectors     5;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       100000;
}

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

It is important to observe that although the solver is implemented with the SIMPLE algorithm, the subdictionary is still named PISO and the keyword nCorrectors will be the number of iterations per time step. Further the keyword pRefCell is set as default to 0 and the pRefValue to $1e5$.

Step 4: Boundary and field set-up

In order to know the created patches in Step 2, we will use paraFoam to select and unselect the regions and visualise the location of the different patches. Doing that and editing the boundary dictionary in constant/polymesh we end with the changes shown below.

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

7
(
sidewall_up
{
    type patch;
    physicalType slip;
    nFaces 240;
    startFace 954584;
}

inlet
{
    type patch;
    physicalType inlet;
    nFaces 240;
    startFace 954824;
}
```

```
sidewall_down
{
    type patch;
    physicalType slip;
    nFaces 240;
    startFace 955064;
}

outlet
{
    type patch;
    physicalType outlet;
    nFaces 240;
    startFace 955304;
}

wing
{
    type wall;
    physicalType wall;
    nFaces 4220;
    startFace 955544;
}

symmetry_left
{
    type patch;
    physicalType slip;
    nFaces 33006;
    startFace 959764;
}

symmetry_right
{
    type patch;
    physicalType slip;
    nFaces 33006;
    startFace 992770;
}
)

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

Yet we've localized the boundaries and renamed them according to their location. We have also set their base and physical types. Left is the field set-up where the boundary conditions are defined. In the 0/ directory we find all the field variables where we can define the primitive types. In order to do this quickly just start FoamX, open the wing3d case, save and close. This will set all the default values to the field variables. Finally open the U file and set the velocity at inlet to (5 0 0).

Step 5: Implementing the transientSimpleOodles solver and adding the liftDrag utility into it

First of all, if you are using OpenFOAM ver 1.4.1-dev then the transientSimpleOodles and the liftDrag utility are included, otherwise follow these steps

to download, implement and compile these into the OpenCFD version 1.4.1.

transientSimpleOodles solver

If you are at chalmers do the following

```
cd $WM_PROJECT_DIR
cp -r --parents applications/solvers/DNSandLES/transientSimpleOodles $WM_PROJECT_US
```

Othewise find the dev version in sourceforge.net Type

```
 svn co https://openfoam-extend.svn.sourceforge.net/svnroot/
openfoam-extend openfoam-extend
```

Once the files are downloaded copy to your project directory as described above. Enter the directory user-1.4.1/applications/sovers/DNSandLES/ and type the following

```
mv transientSimpleOodles transientSimpleOodles_new
cd transientSimpleOodles_new
mv transientSimpleOodles.C transietSimpleOodles_new.C
cd Make
```

In the files file rename all occurensis of transientSimpleOodles to transientSimpleOodles_new and change the $FOAM_APPBIN to $FOAM_USER_APPBIN. In the options file remove the lduSolvers library from the EXE_LIBS. Type wmake and rehash if needed. Now your new solver is ready and excutable.

liftDrag utility

At chalmers type the following

```
cp -r --parents /chalmers/sw/unsup/OpenFOAM/OpenFOAM-1.4.1-dev/
applications/utilities/postProcessing/wall/
liftDrag $WM_PROJECT_USER_DIR/
cp -r --parents /chalmers/sw/unsup/OpenFOAM/OpenFOAM-1.4.1-dev/src/
postProcessing/incompressible/ $WM_PROJECT_USER_DIR
cd $WM_PROJECT_USER_DIR
cd src/postProcessng/incompressible/
emacs Make/files
```

In the files file change the LIB to FOAM_USER_LIBBIN. Here the library for this application will be created. Change also the name of the library from libincompressiblePostProcessing to libliftDrag Now type ./Allwmake in the postProcessing directory. This will remake the library and put it into the user library. Now move on to the application/utilities/postProcessing/wall/liftDrag folder and change the application binary directory to the user one, i.e.

change FOAM_APPBIN to FOAM_USER_APPBIN in the Make/files. Finally in the Make/options Do some changes so that it is like the one below. We have changed the lnInclude library to the one copied to the user directory. In the EXE_LIBS which links different libraries after compiling, we have removed the incompressiblePostProcessing and added the FOAM_USER_LIBBIN directory and the library of liftDrag.

```
EXE_INC = \
    -I$(WM_PROJECT_USER_DIR)/src/postProcessing/incompressible/
lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/transportModels \
    -I$(LIB_SRC)/turbulenceModels \
    -I$(LIB_SRC)/LESmodels \
    -I$(LIB_SRC)/LESmodels/LESdeltas/lnInclude

EXE_LIBS = \
    -lfiniteVolume \
    -L$(FOAM_USER_LIBBIN) \
    -lliftDrag
```

Now open a new file in the solverdirectory and call it computeForces.H. Type the following (this file is included in the tutorial files)

```
//This file is found at the OpenFOAM forum, modified by Srinath Madhavan, Frank M. Boss and Mohammad El-Alti.
{
    const fvPatchList& patches = mesh.boundary();
    vector Uav = vector(1,0,0);

    forAll(patches, patchI)
    {
        if (isType<wallFvPatch>(patches[patchI]))
        {
    scalar Aref = 1.0;
            scalar U_max = 1.0;
    // Make Uav (1,0,0):
    Uav = Uav/mag(Uav);
    // And now, multiply Uav (a vector) with U_max (a scalar) to get the reference velocity.
    Uav = Uav*U_max;

    // This is the output we will see when a solver is running!
            Info<< "\nWall patch = " << patchI << "\n"
                << "Wall patch name = " << patches[patchI].name() << "\n"
                << "Uav = " << Uav << "\n"
                << "Aref = " << Aref << "\n"

    // Uncomment the following line if you are interested in printing out Lref when computing MomentCoefficient.
      //<< "Lref = " << Lref << "\n"

    // Recall that in OpenFoam, we work only with kinematic viscosity (nu) as the entire momentum equation
    // is divided by density (rho). As a result, we need not bother dividing by 'rho' to calculate the
    // dimensionless force (lift/drag) coefficients. This should actually be explained in liftDrag.H
        << "nu = " << nu << "\n"
```

```
                << "DragCoefficient = "   << liftDrag::dragCoefficient(
U,p,nu,patches[patchI].name(),Uav,Aref)<<"\n"

<< "pressureDragCoefficient = "   << liftDrag::pressureDragCoefficient(
U,p,nu,patches[patchI].name(),Uav,Aref)<<"\n"

<< "viscDragCoefficient = "   << liftDrag::viscDragCoefficient(
U,p,nu,patches[patchI].name(),Uav,Aref)<<"\n"

<< "LiftCoefficient = "   << (liftDrag::liftCoefficient(
U,p,nu,patches[patchI].name(),Uav,Aref)& vector(0,1,0))<<"\
    // Uncomment the following line if you are interested in MomentCoefficient. Ensure that Lref is not zero!!!
      //<< "MomentCoefficient = " << (liftDrag::momentCoefficient(
      //U,p,nu,patches[patchI].name(),Uav,Aref,Lref)& vector(0,0,1))<<"\n"
                << endl;

        }
    }
}
```

Create a new file called liftDrag.H in the solver directory. Copy all code in the liftDrag.H found in the src/.../lnInclude/liftDrag.H to the header and then append all the code in src/.../lnInclude/liftDrag.C before #endif. Comment all the turbulence related code so you end with the code attched with the tutorial files.

Finally in the solver file transientSimpleOodles.C add the following code segments. Before main add the following to lines

```
#include "liftDrag.H"
#include "wallFvPatch.H"
```

After #include writeNaveragingSteps.H add the following line

```
#include computeForces.H
```

In the createField.H after the IOdictionary transportProperties block add the following code segment

```
    dimensionedScalar nu
    (
        transportProperties.lookup("nu")
    );
```

Now your new solver is ready for compiling and running.

Step 6: Running the 2D case

Define the boundary conditions according to Step 4, check their locations using paraFoam and run with the icoFoam solver.

Step 7: Mapping and running the 3D case

Now the 2D solution solved with icoFoam will be mapped into our 3D case. Type

```
run
cp $FOAM_TUTORIALS/icoFoam/cavityGrade/system/mapFieldsDict wing3d/system/
```

Edit the mapFieldsDict and add all the pairs of boundaries in the patchMap subdict, see example below.

```
// *******************************************************************//
patchMap
(
   inlet inlet
   outlet outlet
   symm_right symmetry_right
   ...
)
```

Finally type the following in your run directory

```
mapFields . wing2d . wing3d
```

Run now your LES solver with

```
transientSimpleOodles . wing3d > log &
```

7 Active Flow Control

Step 8: Using the patchTool to define a face set

The patchTool is a very powerful tool which makes it possible to view the patches, faces and edges of a model. It also provides the possibility to redefine patches and choose (by the mouse cursor) faces to a new set in order to define a new patch. It also gives you the face number and location cordinates of each chosen boundary face. We will first copy the wing3d directory and call it afc_wing3d. Do as below

```
run
cp -r wing3d afc_wing3d
patchTool . afc_wing3d
```

When the patchTool has started, click connect and load mesh. Go to the *Display* tab and clich on *Create faces* buttom. Finally go to the *Displaylist* tab and unselect axis and all the patches but the wing. See figure 4. You

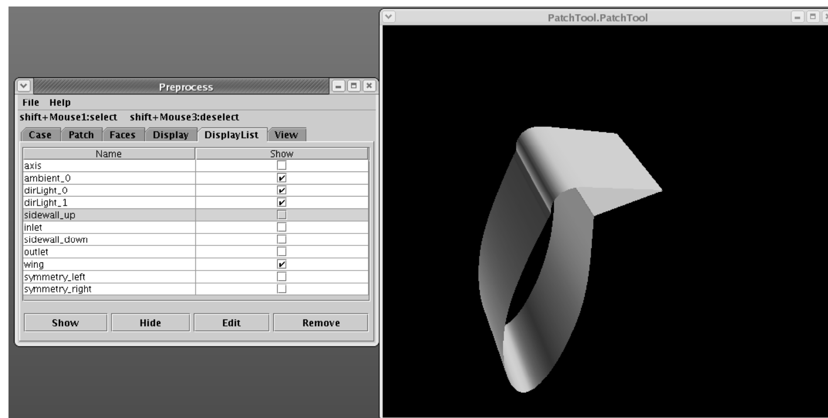

Figure 4: The patchTool

can rotate the model with the left mouse bottom, zoom with the middle one and move with the right one. Go to the *Patch* tab, and add a new patch called actuator as shown in Figure 5. Select it from the patch list and go to the *Faces* tab. Now click the shift bottom and with your mouse left click on any face on the model to select it. Chose some faces on the flap along the z-direction. Unselect with the right bottom on your mouse. Once finished

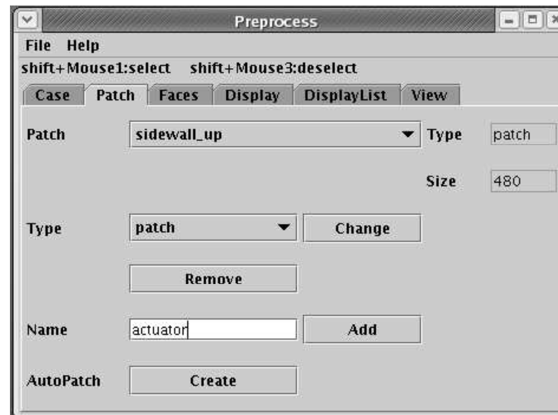

Figure 5: Adding new patch

as Figure 6 click on *change to*-buttom in the *Faces* tab. Finally save the mesh in the *Case* tab. The new mesh is found in the latest time directory, remove or rename the old mesh in constant/polyMesh and move the new polyMesh directory into the constant directory.

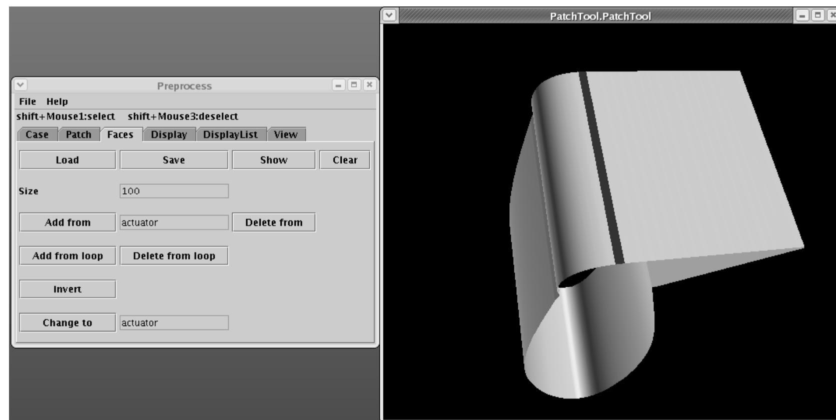

Figure 6: Faces chosen on the model to add to the actuator patch

Step 9: The ocsillatory inlet boundary condition

It is time to set the oscillatoryFixedValue boundary condition. Three parameters are to be set. According to the equation below

$$U = refValue + amplitude \cdot \sin(2\pi \cdot frequency \cdot TIME) \qquad (1)$$

The refValue is a of Field type and should be a vector in this case. The amplitude and the frequency are scalars. So set the boundary condition of actuator for U field like below

```
actuator
{
    type            oscillatingFixedValue;
    refValue        uniform (2 2 0);
    amplitude       10;
    frequency       10;
}
```

8 Postprocessing

Step 10: Plot the drag, velocity and pressure field

When the simulation is done. Use paraFoam to plot the velocity and pressure field.

A Notes on the transientSimpleOodles solver

The transientSimpleOodles solver is a copy of the original oodles solver with the only difference of implemented solution algorithm. The former one uses SIMPLE and the latter one uses PISO. The filtered Navier-Stokes reads

$$\frac{\partial \bar{u}_i}{\partial t} + \frac{\partial}{\partial t}(\bar{u}_i \bar{u}_j) = -\frac{1}{\rho}\frac{\partial \bar{p}}{\partial x_i} + \nu \frac{\partial^2 \bar{p}}{\partial x_j \partial x_j} - \frac{\partial \tau}{\partial x_j} \tag{2}$$

where τ_{ij} is the subgrid scale stresses modeled as

$$\tau_{ij} - 1/3\delta_{ij}\tau kk = -2\nu_{sgs}\bar{s}_{ij} \tag{3}$$

where ν_{sgs} is

$$\nu_{sgs} = (C_S\Delta)^2\sqrt{2\bar{s}_{ij}\bar{s}_{ij}} \tag{4}$$

and $\bar{s}_{ij}$ is

$$\bar{s}_{ij} = 1/2\left(\frac{\partial \bar{u}_j}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right) \tag{5}$$

The souce code for the transientSimpleSolver begin with the name of the application and a summary description of it.

```
Application
    transientSimpleOodles

Description
    Incompressible LES solver using transient SIMPLE.

\*---------------------------------------------------------------------------*/

#include "fvCFD.H"
#include "incompressible/singlePhaseTransportModel/singlePhaseTransportModel.H"
#include "incompressible/transportModel/transportModel.H"
#include "incompressible/LESmodel/LESmodel.H"
#include "IFstream.H"
#include "OFstream.H"
#include "Random.H"

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

The description follows with a lot of include statements where the solver resources the CFD tools single phase transport model and LES models.

The code continue with the main

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

int main(int argc, char *argv[])
{

#   include "setRootCase.H"
#   include "createTime.H"
#   include "createMeshNoClear.H"
#   include "createFields.H"
#   include "createAverages.H"
#   include "initContinuityErrs.H"


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

Where the case is initialised by setting the root case, time, fields, averages and continuity errors.

The code continues by printing "Starting time loop", and the time loop starts.

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

    Info<< "\nStarting time loop\n" << endl;

    for (runTime++; !runTime.end(); runTime++)
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

#       include "readPISOControls.H"
#       include "CourantNo.H"

        sgsModel->correct();
```

Inside the time loop the code reads the algorithm control variables and prints the current courant number and corrects the SGS model. Now the SIMPLE loop begins and the equation solved.

```
        // Pressure-velocity SIMPLE corrector
        for (int corr=0; corr<nCorr; corr++)
        {
            // Momentum predictor

            tmp<fvVectorMatrix> UEqn
            (
                fvm::ddt(U)
              + fvm::div(phi, U)
              + sgsModel->divB(U)
            );

            UEqn().relax();

            solve(UEqn() == -fvc::grad(p));

            p.boundaryField().updateCoeffs();
            volScalarField rUA = 1.0/UEqn().A();
            U = rUA*UEqn().H();
            UEqn.clear();
            phi = (fvc::interpolate(U) & mesh.Sf())
                + fvc::ddtPhiCorr(rUA, U, phi);

            adjustPhi(phi, U, p);


            // Non-orthogonal pressure corrector loop
            p.storePrevIter();

            for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
            {
                fvScalarMatrix pEqn
                (
                    fvm::laplacian(rUA, p) == fvc::div(phi)
                );

                pEqn.setReference(pRefCell, pRefValue);
```

```
                pEqn.solve();

                if (nonOrth == nNonOrthCorr)
                {
                    phi -= pEqn.flux();
                }
            }

#           include "continuityErrs.H"

            // Explicitly relax pressure for momentum corrector
            p.relax();

            // Momentum corrector
            U -= rUA*fvc::grad(p);
            U.correctBoundaryConditions();
        }
```

Finally the calculations of averages are done and some information about the run are printed out.

```
#       include "calculateAverages.H"

        runTime.write();

#       include "writeNaveragingSteps.H"
//Here we include the computeForces.H file in order to calculate
//and print out the force coefficients
# include "computeForces.H"

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }

    Info<< "End\n" << endl;

    return(0);
}
```