CFD with Open Source Software, Final Assignment

Python Scripting and M4-Scripting for automatization and parameterization in OpenFOAM

Author :
Erwin Adi Hartono

Reviewer :
Alireza Javidi
Jelena Andric

Content

Chapter 1
Introduction

1.1. Background

The goal of this project is to optimize the curve of contraction section of open circuit wind tunnel. The wind tunnel contraction is an important part that determine the flow quality in the test section. The main effect of a contraction are to reduce both mean and fluctuating velocity fluctuation to a smaller fraction of the average velocity and to increase the flow mean velocity.[1]

To achieve this goal, some curves have to be selected and to be tested. This is obviously a repetition work, and it is more convenient to make it automatic. Automatization can be done by using phyton script. The computer will happily do what is written in that script. With this approach, a repetition work is done easily.

Different contraction means different geometry. The `blockmeshdict` has to be modified every time we have a new contraction. It is more convinient to parameterize the `blockmeshdict`. Parameterization of `blockmeshdict` is done by using m4-scripting.

1.2. The Geometry and The Mesh

OpenFOAM always operates in a 3 dimensional Cartesian coordinate system and all geometries are generated in 3 dimensions. OpenFOAM solves the case in 3 dimensions by default but can be instructed to solve in 2 dimensions by specifying a 'special' empty boundary condition on boundaries normal to the (3rd) dimension for which no solution is required.[6]

The geometry of an open circuit wind tunnel is relatively easy to make in `blockMeshDict`, especially because this is just a 2D calculation. Figure 1 shows the basic geometry of the wind tunnel. The analysis will only be done at half piece of the tunnel. The z-direction in figure 1 is exaggerated for the clarity in viewing the vertices.

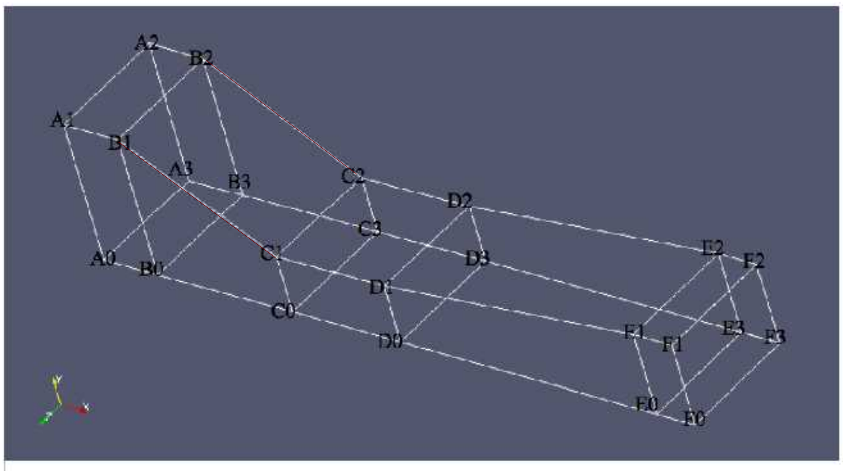

Figure 1. Basic Geometry and Vertices

Each edge joining 2 vertex points is assumed to be straight by default. However any edge may be specified to be curved by entries in a list named `edges`. Each entry for a curved edge begins with a keyword specifying the type of curve from those listed in Table 1. The keyword is then followed by the labels of the 2 vertices that the edge connects. Following that, interpolation points must be specified through which the edge passes.[7]

Keyword selection	Description	Additional entries
`arc`	Circular arc	Single interpolation point
`simpleSpline`	Spline curve	List of interpolation points
`polyLine`	Set of lines	List of interpolation points
`polySpline`	Set of splines	List of interpolation points
`line`	Straight line	—

Table 1. Edge types available in the blockMeshDict dictionary

In this project, the edges B1-C1 and B2-C2 will be shaped into curve. The curve will follow the equation that has been developed by Daniel Brassard[3]. The equation is :

$$h=[\eta(H_i^{1/\alpha}-H_o^{1/\alpha})+H_i^{1/\alpha}]^{\alpha} \qquad (1)$$

$$\eta=-10\xi^3+15\xi^4-6\xi^5 \qquad (2)$$

$$\xi=\frac{x}{L} \qquad (3)$$

x	= the coordinate in x-direction
L	= total length of contraction (B0-C0)
H_i	= Contraction inlet height (B0-B1)
H_o	= Contraction outlet height (C0-C1)
α	= a function of ξ, defined for $0<\xi<1$

A simple plot has been done in MATLAB to see the shape of the curve that is produced by the equation above. Figure 2 shows the variations of the curve with different alpha value.

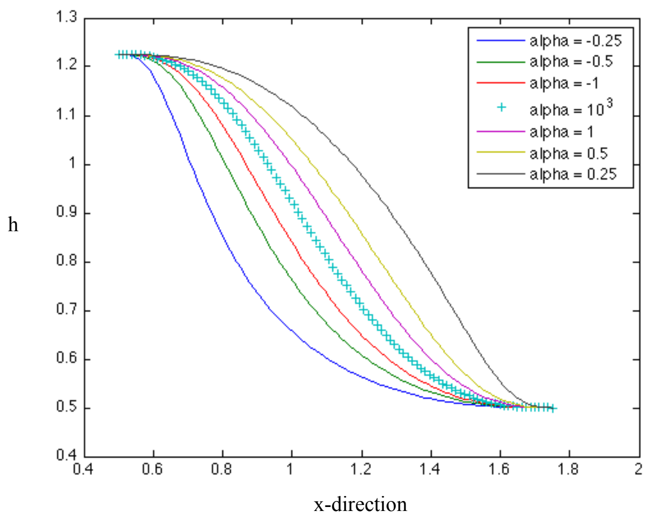

figure 2. Plot of contraction curve in different alpha value

All parameters in the equation above are known except the alpha value. This alpha value can be anything, can be a constant or a function, and still give the smooth curve.[3] In this project alpha value that vary linearly from -1 to 1 is going to be tested. For simplicity, MATLAB plot is made to see the shape of the curve and then three value that represent the most extreme shape is chosen.

The alpha value employed in this project is -0.25, +0.25 and 10^3. The reason why these three numbers are chosen are alpha = -0.25 gives a curve that has small radius upstream, alpha = 0.25 gives a curve that has small radius downstream. Alpha=10^3 is chosen because it provides a curve with almost equivalent upstream/downstream radius.

`simpleSpline` command will be used to make the curve. The syntax is:

```
simpleSpline startVertex endVertex
             (list of interpolation points)
```

In `Blockmeshdict.m4` will look like this :

```
edges
(
       simpleSpline B1 C1
       (
       (x1 y1 0)
       (x2 y2 0)
       (x3 y3 0)
       (x4 y4 0)
       (x5 y5 0)
       (x6 y6 0)
       (x7 y7 0)
       (x8 y8 0)
       (x9 y9 0)
       (x10 y10 0)
       )

       simpleSpline B2 C2
       (
       (x1 y1 z)
       (x2 y2 z)
       (x3 y3 z)
       (x4 y4 z)
       (x5 y5 z)
       (x6 y6 z)
       (x7 y7 z)
       (x8 y8 z)
       (x9 y9 z)
       (x10 y10 z)
       )
);
```

1.3. Boundary Condition

In open circuit wind tunnel, air is sucked by a fan that is located at the back of the tunnel. For simplicity of this project the air 10 m/s is blown uniformly from forward of the tunnel.

The simplest way to set the boundary condition for this case is just simply copy the `0` directory from `pitzDaily` case. It's because the boundary condition is the same as `pitzDaily case.`

1.4. Solver

The calculation is in 2D and using `simpleFoam` solver to analyze the flow and `calcPressureDifference` utilities for calculating the pressure difference between inlet and outlet of wind tunnel to find the best contraction design in term of pressure difference.

Chapter 2
Setting Up The Case

2.1. blockmeshdict.m4

blockMeshDict is built from blockMeshDict.m4. M4-scripting is used to parmeterized the blockMeshDict so we can easily manipulate the blockMeshDict.

All definitions are done before the OpenFOAM header. Inside blockmeshdict the syntax is all the same, but the numbers now become variable that has been defined previously.

Some definitions of m4-script command :

command	description
m4	run the script
m4_define	define variable, command, value

This is inside blockmeshdict.m4.

```
//Parametrized windTunnel
//Run using:
//m4 -P blockMeshDict.m4 > blockMeshDict
//m4 definitions:
m4_changecom(//)m4_changequote([,])
m4_define(calc, [m4_esyscmd(perl -e 'use Math::Trig; printf ($1)')])
m4_define(VCOUNT, 0)
m4_define(vlabel, [[// ]Vertex $1 = VCOUNT m4_define($1, VCOUNT)m4_define([VCOUNT],
m4_incr(VCOUNT))])

//Mathematical constants:
m4_define(pi, 3.1415926536)

//m4 spec:
m4_define(testSectionInletWidth, 0.5)
m4_define(testSectionInletHeight, testSectionInletWidth)
m4_define(testSectionLength, calc(2*testSectionInletWidth))
m4_define(testSectionOutletWidth, testSectionInletWidth)
m4_define(testSectionOutletHeight, testSectionInletHeight)
m4_define(testSectionInletArea, calc(testSectionInletWidth*testSectionInletHeight))
m4_define(contractionRatio, 6)
m4_define(contractionInletArea, calc(contractionRatio*testSectionInletArea))
m4_define(contractionInletHeight, calc(sqrt(contractionInletArea)))
m4_define(contractionInletWidth, contractionInletHeight)
m4_define(contractionOutletWidth, testSectionInletWidth)
m4_define(contractionOutletHeight, testSectionInletHeight)
m4_define(contractionLength, calc(2.5*contractionOutletHeight))
m4_define(settlingChamberLength, testSectionInletWidth)
m4_define(settlingChamberOutletWidth, contractionInletWidth)
m4_define(settlingChamberOutletHeight, contractionInletWidth)
m4_define(settlingChamberInletWidth, settlingChamberOutletWidth)
m4_define(settlingChamberInletHeight, settlingChamberOutletWidth)
m4_define(diffuserRatio, 2)
m4_define(diffuserAngle, 2.5)
m4_define(diffuserInletWidth, testSectionOutletWidth)
m4_define(diffuserInletHeight, testSectionOutletWidth)
m4_define(diffuserOutletArea, calc(diffuserRatio*diffuserInletWidth*diffuserInletHeight))
m4_define(diffuserOutletHeight, calc(sqrt(diffuserOutletArea)))
m4_define(diffuserOutletWidth, diffuserOutletHeight)
m4_define(diffuserLength, calc(0.5*diffuserOutletHeight/tan(diffuserAngle*pi/180)-
0.5*diffuserInletHeight/tan(diffuserAngle*pi/180)))
m4_define(fanInletHeight, diffuserOutletHeight)
```

```
m4_define(fanInletWidth, diffuserOutletWidth)
m4_define(fanOutletHeight, fanInletHeight)
m4_define(fanOutletWidth, fanInletWidth)
m4_define(fanLength, calc(0.5*fanInletHeight))


//plane A settlingChamberInlet
m4_define(xA, 0)

//plane B contractionInlet
m4_define(xB, calc(xA+settlingChamberLength))

//plane C testSectionInlet
m4_define(xC, calc(xB+contractionLength))

//plane D diffuserInlet
m4_define(xD, calc(xC+testSectionLength))

//plane E fanInlet
m4_define(xE, calc(xD+diffuserLength))

//plane F fanOutlet
m4_define(xF, calc(xE+fanLength))

//plane z:
m4_define(z, 0.05)

//spline:
m4_define(alpha, m4_include(currentAlpha))
m4_define(alphaInv, calc(1/alpha) )
m4_define(L, contractionLength)
m4_define(deltaX, calc(contractionLength/10))
m4_define(deltaH, calc(contractionInletHeight**alphaInv - contractionOutletHeight**alphaInv) )
m4_define(Hi, calc(contractionInletHeight**alphaInv) )

m4_define(x1, xB)
m4_define(x2, calc(xB+1*deltaX))
m4_define(x3, calc(xB+2*deltaX))
m4_define(x4, calc(xB+3*deltaX))
m4_define(x5, calc(xB+4*deltaX))
m4_define(x6, calc(xB+5*deltaX))
m4_define(x7, calc(xB+6*deltaX))
m4_define(x8, calc(xB+7.5*deltaX))
m4_define(x9, calc(xB+8.5*deltaX))
m4_define(x10, xC)

m4_define(x2norm, calc(calc(x2-xB)/L))
m4_define(x3norm, calc(calc(x3-xB)/L))
m4_define(x4norm, calc(calc(x4-xB)/L))
m4_define(x5norm, calc(calc(x5-xB)/L))
m4_define(x6norm, calc(calc(x6-xB)/L))
m4_define(x7norm, calc(calc(x7-xB)/L))
m4_define(x8norm, calc(calc(x8-xB)/L))
m4_define(x9norm, calc(calc(x9-xB)/L))
m4_define(x2norm3, calc(x2norm**3))
m4_define(x3norm3, calc(x3norm**3))
m4_define(x4norm3, calc(x4norm**3))
m4_define(x5norm3, calc(x5norm**3))
m4_define(x6norm3, calc(x6norm**3))
m4_define(x7norm3, calc(x7norm**3))
m4_define(x8norm3, calc(x8norm**3))
m4_define(x9norm3, calc(x9norm**3))
m4_define(x2norm4, calc(x2norm**4))
m4_define(x3norm4, calc(x3norm**4))
m4_define(x4norm4, calc(x4norm**4))
m4_define(x5norm4, calc(x5norm**4))
m4_define(x6norm4, calc(x6norm**4))
m4_define(x7norm4, calc(x7norm**4))
m4_define(x8norm4, calc(x8norm**4))
m4_define(x9norm4, calc(x9norm**4))
m4_define(x2norm5, calc(x2norm**5))
m4_define(x3norm5, calc(x3norm**5))
m4_define(x4norm5, calc(x4norm**5))
m4_define(x5norm5, calc(x5norm**5))
m4_define(x6norm5, calc(x6norm**5))
m4_define(x7norm5, calc(x7norm**5))
m4_define(x8norm5, calc(x8norm**5))
```

```
m4_define(x9norm5, calc(x9norm**5))

m4_define(eta2, calc( -10*x2norm3 +15*x2norm4 -6*x2norm5) )
m4_define(eta3, calc( -10*x3norm3 +15*x3norm4 -6*x3norm5) )
m4_define(eta4, calc( -10*x4norm3 +15*x4norm4 -6*x4norm5) )
m4_define(eta5, calc( -10*x5norm3 +15*x5norm4 -6*x5norm5) )
m4_define(eta6, calc( -10*x6norm3 +15*x6norm4 -6*x6norm5) )
m4_define(eta7, calc( -10*x7norm3 +15*x7norm4 -6*x7norm5) )
m4_define(eta8, calc( -10*x8norm3 +15*x8norm4 -6*x8norm5) )
m4_define(eta9, calc( -10*x9norm3 +15*x9norm4 -6*x9norm5) )

m4_define(y1, contractionInletHeight)
m4_define(y2, calc( calc(eta2*deltaH+Hi)**alpha) )
m4_define(y3, calc( calc(eta3*deltaH+Hi)**alpha) )
m4_define(y4, calc( calc(eta4*deltaH+Hi)**alpha) )
m4_define(y5, calc( calc(eta5*deltaH+Hi)**alpha) )
m4_define(y6, calc( calc(eta6*deltaH+Hi)**alpha) )
m4_define(y7, calc( calc(eta7*deltaH+Hi)**alpha) )
m4_define(y8, calc( calc(eta8*deltaH+Hi)**alpha) )
m4_define(y9, calc( calc(eta9*deltaH+Hi)**alpha) )
m4_define(y10, contractionOutletHeight)


/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.7.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;

    root            "";
    case            "";
    instance        "";
    local           "";

    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

vertices
(
       // front side
       (xA 0 0) vlabel(A0)
       (xB 0 0) vlabel(B0)
       (xC 0 0) vlabel(C0)
       (xD 0 0) vlabel(D0)
       (xE 0 0) vlabel(E0)
       (xF 0 0) vlabel(F0)
       (xF fanOutletHeight 0) vlabel(F1)
       (xE diffuserOutletHeight 0) vlabel(E1)
       (xD testSectionOutletHeight 0) vlabel(D1)
       (xC contractionOutletHeight 0) vlabel(C1)
       (xB settlingChamberOutletHeight 0) vlabel(B1)
       (xA settlingChamberInletHeight 0) vlabel(A1)

       // back side
       (xA 0 z) vlabel(A3)
       (xB 0 z) vlabel(B3)
       (xC 0 z) vlabel(C3)
       (xD 0 z) vlabel(D3)
       (xE 0 z) vlabel(E3)
       (xF 0 z) vlabel(F3)
       (xF fanOutletHeight z) vlabel(F2)
       (xE diffuserOutletHeight z) vlabel(E2)
       (xD testSectionOutletHeight z) vlabel(D2)
       (xC contractionOutletHeight z) vlabel(C2)
       (xB settlingChamberOutletHeight z) vlabel(B2)
       (xA settlingChamberInletHeight z) vlabel(A2)
```

```
);

blocks
(
    // settlingChamber
    hex (B3 A3 A2 B2 B0 A0 A1 B1) (10 40 1) simpleGrading (1 0.05 1)
    // contraction
    hex (C3 B3 B2 C2 C0 B0 B1 C1) (50 40 1) simpleGrading (1 0.05 1)
    // testSection
    hex (D3 C3 C2 D2 D0 C0 C1 D1) (50 40 1) simpleGrading (1 0.05 1)
    // diffuser
    hex (E3 D3 D2 E2 E0 D0 D1 E1) (60 40 1) simpleGrading (1 0.05 1)
    // fan
    hex (F3 E3 E2 F2 F0 E0 E1 F1) (10 40 1) simpleGrading (1 0.05 1)
);

edges
(
        simpleSpline B1 C1
        (
        (x1 y1 0)
        (x2 y2 0)
        (x3 y3 0)
        (x4 y4 0)
        (x5 y5 0)
        (x6 y6 0)
        (x7 y7 0)
        (x8 y8 0)
        (x9 y9 0)
        (x10 y10 0)
        )

        simpleSpline B2 C2
        (
        (x1 y1 z)
        (x2 y2 z)
        (x3 y3 z)
        (x4 y4 z)
        (x5 y5 z)
        (x6 y6 z)
        (x7 y7 z)
        (x8 y8 z)
        (x9 y9 z)
        (x10 y10 z)
        )


);

patches
(
        patch inlet
        (
        (A3 A0 A1 A2)
        )

        patch outlet
        (
        (F3 F0 F1 F2)
        )

        wall upperWall
        (
        (B2 B1 A1 A2)
        (C2 C1 B1 B2)
        (D2 D1 C1 C2)
        (E2 E1 D1 D2)
        (F2 F1 E1 E2)
        )

        symmetryPlane axis
        (
        (B3 B0 A0 A3)
        (C3 C0 B0 B3)
        (D3 D0 C0 C3)
        (E3 E0 D0 D3)
        (F3 F0 E0 E3)
        )

        empty frontAndBack
        (
        (B0 A0 A1 B1)
        (C0 B0 B1 C1)
        (D0 C0 C1 D1)
        (E0 D0 D1 E1)
        (F0 E0 E1 F1)
        (A3 B3 B2 A2)
        (B3 C3 C2 B2)
        (C3 D3 D2 C2)
        (D3 E3 E2 D2)
        (E3 F3 F2 E2)
        )
);

mergePatchPairs
(
);

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

2.2. calcPressureDifference

This utility is implemented by Bernhard Gschaider who has also developed `pyFoam`. The detail of this utilities can be found in `http://openfoamwiki.net/index.php/Contrib_calcPressureDifference`

Unfortunately, due to the fact that this utility is developed for OpenFOAM.v1.2, the directory order is a little bit different, so to make it works the `options` file in `Make` directory has to be modified.

Copy paste the lines below into the `options` file and do `wmake`.

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude

EXE_LIBS = \
    -lfiniteVolume
```

`calcPressureDifference` needs `calcPressureDifferenceDict` to work. So, copy the `calcPressureDifferenceDict` to the `system` directory.

2.3. erwin.py

`erwin.py` is basically just a name, the file name can be anything as long as at the first line of the script is this statement, `#!/usr/bin/python` This statement will tell the computer that this script is a python script.

Inside the script there is a loop that will:

1. copy the `baseline` directory and name it with `modified<alphaValue>`,
2. change the alpha value inside `currentAlpha` file,
3. run the `m4-script` to make the `blockMeshDict`,
4. create `blockMesh`,
5. solved the case with `simpleFoam` and save the output into `simpleFoamLog`,
6. calculate the pressure difference between inlet and outlet and save the output into `calcPressureDifferenceLog`, and
7. it will go over with the different alpha value and do the same procedure of solving.

This is inside `erwin.py` :

```
#!/usr/bin/python


import sys, os, shutil, math, commands, glob

from pylab import *

from PyFoam.Execution.UtilityRunner import UtilityRunner
from PyFoam.Execution.BasicRunner import BasicRunner
from PyFoam.RunDictionary.SolutionDirectory import SolutionDirectory
from PyFoam.RunDictionary.BlockMesh import BlockMesh

from os import path

from subprocess import Popen
from subprocess import call

orig=SolutionDirectory(path.expandvars("$FOAM_RUN/windTunnel/baseline"),archive=None,paraviewLink=Fa
lse)

alphas = [-0.25, 0.25, 1000]

for alpha in alphas:

        print "alpha:",alpha
        # clone the original case
        case=orig.cloneCase("modified%f" % alpha).name

        print "m4 -P blockMeshDict.m4 > blockMeshDict"
        # cd into case directory
        os.chdir(case)

        # change the alpha value inside currentAlpha
        infilename='constant/polyMesh/currentAlpha'
        outfilename='constant/polyMesh/currentAlphaTemp'
        ifile = open( infilename, 'r')
        ofile = open(outfilename, 'w')
        lines = ifile.readlines()
        ofile.write(str(alpha))
        ifile.close()
        ofile.close()
        os.remove(infilename)
        os.rename(outfilename,infilename)

        # run m4
        os.chdir("constant/polyMesh")
        cmd='m4 -P blockMeshDict.m4 > blockMeshDict'
        pipefile = open('output', 'w')
        retcode = call(cmd,shell=True,stdout=pipefile)
        pipefile.close()
        os.remove('output')
        os.chdir("../../..")

        # run blockMesh
        print "blockMesh"
        os.chdir(case)
        cmd='blockMesh'
        pipefile = open('output', 'w')
        retcode = call(cmd,shell=True,stdout=pipefile)
        pipefile.close()
        os.remove('output')
        os.chdir("..")

        # run solver
        print "simpleFoam"
        os.chdir(case)
        cmd='simpleFoam > simpleFoamLog'
        pipefile = open('output', 'w')
        retcode = call(cmd,shell=True,stdout=pipefile)
        pipefile.close()
        os.remove('output')
        os.chdir("..")
```

```
# calculate the pressure difference
print "calcPressureDifference"
os.chdir(case)
cmd='calcPressureDifference > calcPressureDifferenceLog'
pipefile = open('output', 'w')
retcode = call(cmd,shell=True,stdout=pipefile)
pipefile.close()
os.remove('output')
os.chdir("..")
```

Chapter 3
Running The Case

Step by step to run the `windTunnel` case:

1. Copy the `windTunnel` directory into `run` directory. Inside that directory there is another directory called `baseline` and a python script, `erwin.py`
2. Change directory in the terminal window into `windTunnel` directory
3. Run the python script by typing `./erwin.py`
4. Analyse the results by opening all modified directories

Chapter 4
The Result

This chapter shows the result from the solver. These are screenshots of the last time step of pressure and velocity distribution. There is no analysis has been made for this problem, because it is outside the scope of this course. However, the good contraction will lead to the uniformity of flow and thin boundary layer at the test section.

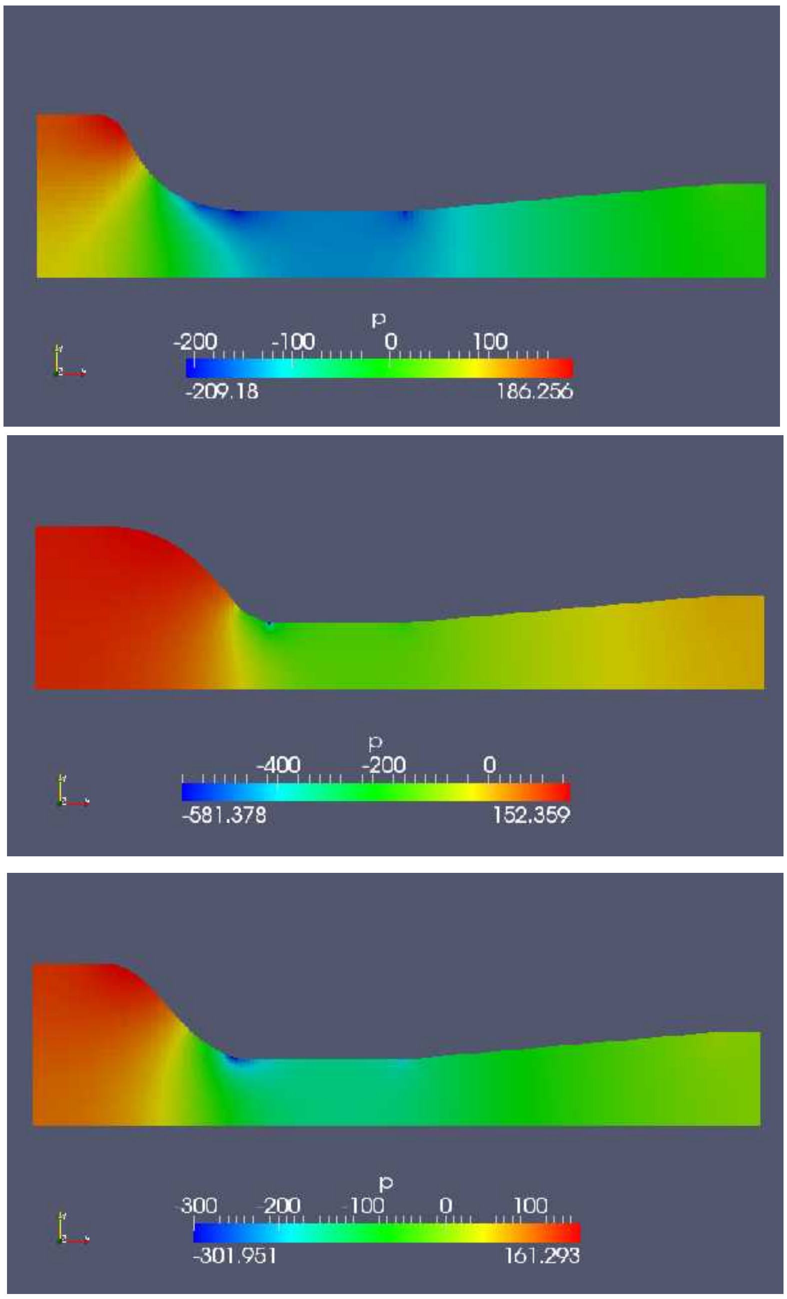

figure 3. pressure distribution for alpha = -0.25, 0,25 and 10^3

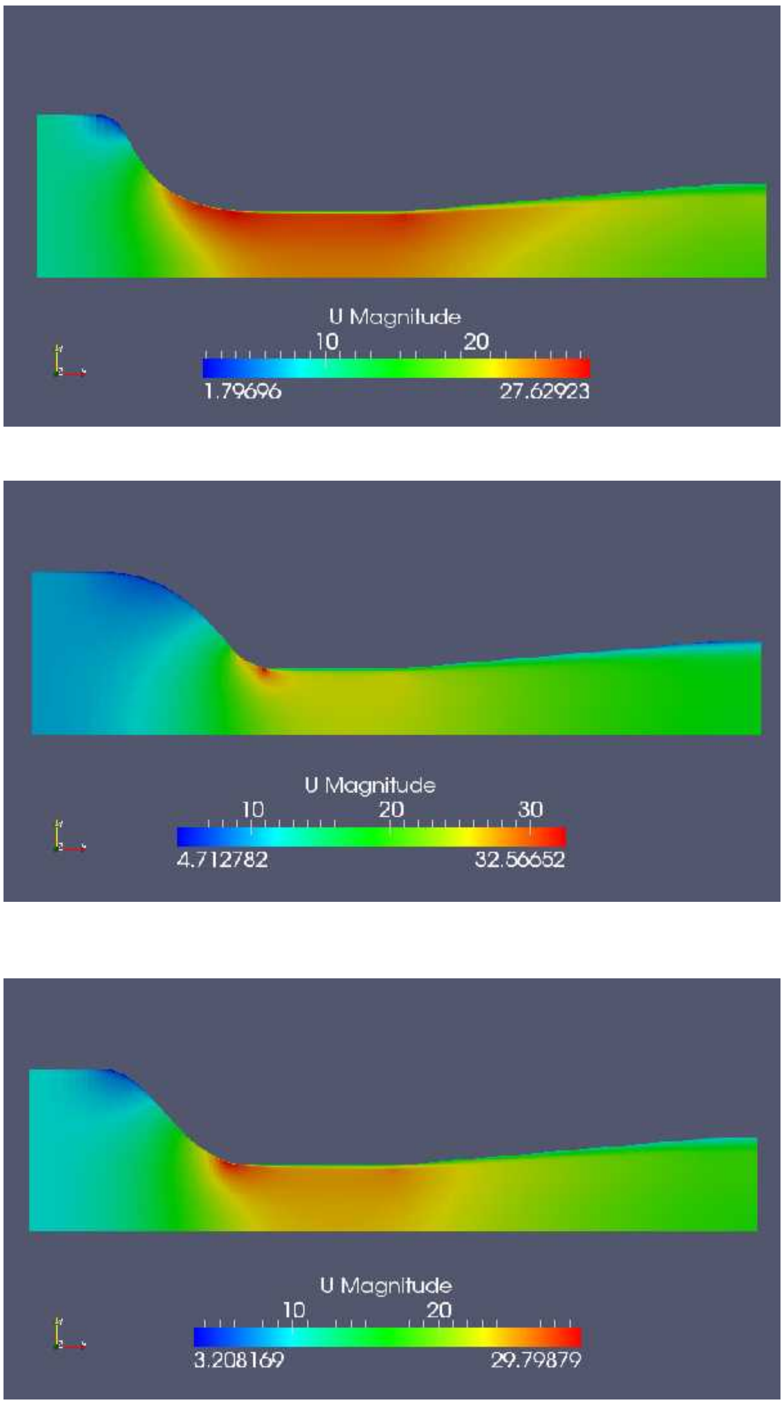

figure 4. pressure distribution for alpha = -0.25, 0,25 and 10^3

References

[1] Bell, J. and Mehta, R., “Boundary Layer Predictions for Small Low Speed Contractions,” AIAA Journal, Vol. 27, no. 3, March 1989, pp. 372-374.

[2] Mehta, R. and Bradshaw, P., “Design Rules for Small Low Speed Wind Tunnels,” Aeronautical Journal , Vol. 73, 1979, pp. 443–449.

[3] Brassard, D and Ferchichi, M., “Transformation of a polynomial for a contraction wall profile”., J. Fluid. Eng., 127 (1), 183-185, 2005.

[4] Gschaider, B.F.W., “Happy Foaming with Python”, 4th OpenFoam Workshop. 2009

[5] Gschaider, B.F.W., “Automatization with pyFoam, How Python Helps Us Avoid Contact with OpenFOAM”, 5th OpenFoam Workshop. 2010

[6] Paterson, E.G., “Python Scripting for Gluing CFD Applications: A Case Study Demonstrating Automation of Grid Generation, Parameter Variation, Flow Simulation, Analysis, and Plotting”, Technical Report No. TR 09-001,The Pennsylvania State University, The Applied Research Laboratory, 13 January 2009.

[7] OpenFOAM., “OpenFOAM, The Open Source CFD Toolbox, User Guide”, Version 1.6. , July 2009.