Final: Incompressible, Laminar Flow over a Rectangular Cavity

John Karasinski
Graduate Student Researcher
Center for Human/Robotics/Vehicle Integration and Performance
Department of Mechanical and Aerospace Engineering
University of California
Davis, California 95616
Email: karasinski@ucdavis.edu

1 Problem Description

Forty five years ago, Mehta and Lavan published a paper on the numerical investigation of flow over a rectangular cavity at low Reynolds numbers [1]. This relatively simple geometry provides tremendous insight into the physics of flow separation, an important flow feature in many applications. A numerical 2D planar model of these incompressible, laminar flows is developed here. In particular, the predicted flow structure (streamline pattern, eddies) and velocity profiles are investigated for a variety of aspect ratios (AR) and Reynolds numbers (Re).

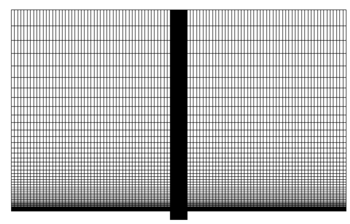

Fig. 1: Generated grid for the AR=0.5 cases

2 Numerical Solution Approach

The pisoFoam solver from OpenFOAM 2.3.0 was used to model the solution of this problem. pisoFoam is a transient solver for incompressible flow which supports multiple forms of turbulence modelling. A Reynolds-average simulation (RAS) turbulence model is employed with a standard $k-\varepsilon$ model. The solver is initialized with the initial conditions described in Table 1. Additionally, the boundary field for the inlet and outlet boundaries are set to "zeroGradient" for all of the initial fields and, the boundary field for frontAndBack is set to "empty" for all initial fields, turning this into a 2D problem.

Five cases were investigated: a base case with AR=0.5 and Reynolds number (Re) of 100, and additional cases of AR=0.5 with Re=1 and 2000 for AR=0.5, and AR=2.0 and 5.0 for Re=100. A Python script was created to generate the initial conditions and geometry for each case. The script generated the nonuniform grid using the blockMesh tool. The resultant mesh for the base case can be see in Figure 2. The domain of the problem was then split on to four CPUs using the decomposePar tool, and the pisoFoam solver was called with the MPI option. The solver than solved the system for 360 seconds and reconstructed the domain using the reconstructPar tool.

3 Results Discussion

4 Conclusion

References

[1] Mehta, U. B., and Lavan, Z., 1969. "Flow in a two-dimensional channel with a rectangular cavity". *Journal of Applied Mechanics,* **36**(4), pp. 897–901.

	internal	lid	fixedWalls
U [m/s]	(0 0 0)*	(1 0 0)*	(0 0 0)*
p [m^2/s^2]	0*	zeroGradient	zeroGradient
ε [m^2/s^3]	0.000765*	0.000765*	0.000765*
k [m^2/s^2]	0.00325*	0.00325*	0.00325*
ν_t [m^2/s]	0*	0*	0*

Table 1: Initial conditions for simulation (*: uniform fields)

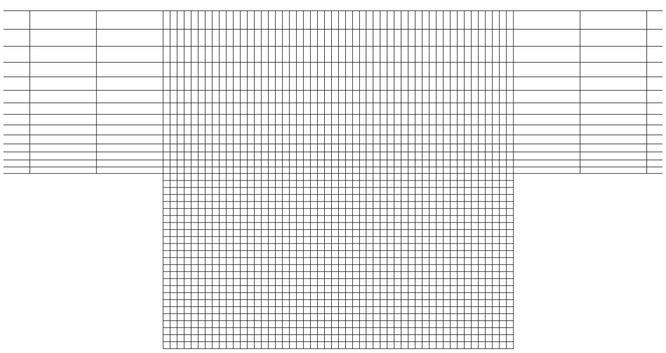

Fig. 2: Closeup on the cavity generated grid for the AR=0.5 cases

Appendix A: Python Code

```
import subprocess
import os
from PrettyPlots import *


def inplace_change(filename, old_string, new_string):
    with open(filename) as f:
        s = f.read()

    if old_string in s:
        # print 'Changing "{old_string}" to "{new_string}"'.format(**locals())
        s = s.replace(old_string, new_string)
        with open(filename, 'w') as f:
            f.write(s)
    # else:
        # print 'No occurances of "{old_string}" found.'.format(**locals())


def subprocess_cmd(command):
    process = subprocess.Popen(command, stdout=subprocess.PIPE, shell=True)
    proc_stdout = process.communicate()[0].strip()
    # print proc_stdout


def generate_folders(ARs, Res):
    for AR, Re in zip(ARs, Res):
        run = "Run" + str(AR) + '-' + str(Re)
        if not os.path.exists(run):
            command = "cp -rf base/ " + run + "/; "
            subprocess_cmd(command)

    print ('Folders generated.')


def create_mesh_file(path, AR, Re):
    mesh = 50.  # where 40 is a mesh size that gives results at the necessary spacing

    YMESH    = str(int(mesh * AR))
    INV_MESH = str(1. / mesh)
    MESH     = str(int(mesh))
    AR       = str(-AR)

    inplace_change(path, 'AR',             AR)
    inplace_change(path, 'XMESH',       MESH)
    inplace_change(path, 'YMESH',       YMESH)
    inplace_change(path, 'INV_MESH', INV_MESH)
    inplace_change(path, 'MESH',         MESH)


def create_properties_file(path, AR, Re):
    d = 12.           # depth of chasm
    NU = str(d / Re)

    inplace_change(path, 'NU', NU)


def update_dimensions(ARs, Res):
    for AR, Re in zip(ARs, Res):
        run = "Run" + str(AR) + '-' + str(Re)
        path = run + '/constant/polyMesh/blockMeshDict'
        create_mesh_file(path, AR, Re)
        # with open(path, 'w') as config_file:
            # config_file.write(create_mesh_file(AR, Re))

        path = run + '/constant/transportProperties'
        create_properties_file(path, AR, Re)
        # with open(path, 'w') as config_file:
```

```
            # config_file.write(create_properties_file(AR, Re))

    print ('Config generated.')


def run_simulations(ARs, Res):
    for AR, Re in zip(ARs, Res):
        run = "Run" + str(AR) + '-' + str(Re)
        if not os.path.exists(run + '/log'):
            print(run + ' running now.')
            command = "hdiutil attach -quiet -mountpoint $HOME/OpenFOAM OpenFOAM.sparsebundle; "
            command += "sleep 1; "
            command += "source $HOME/OpenFOAM/OpenFOAM-2.3.0/etc/bashrc; "
            command += "cd " + run + "; "
            command += "blockMesh; "
            command += "decomposePar; "
            command += "mpirun -np 4 pisoFoam -parallel > log; "
            command += "reconstructPar; "
            command += "streamFunction; "
            # command += 'paraFoam --script="../paraFoam.py" '

            subprocess_cmd(command)
        print(run + ' complete.')

    print('Simulations complete.')


def main(ARs, Res):
    print('Running ARs ' + str(ARs) + ' with Res ' + str(Res) + '.')
    generate_folders(ARs, Res)
    update_dimensions(ARs, Res)
    run_simulations(ARs, Res)
    # generate_plots(ARs, Res)
    print('Done!')

if __name__ == "__main__":
    # Base case
    ARs = [  0.5]
    Res = [100.0]
    #          o                       Broken=x Working=o
    main(ARs, Res)

    # Additional cases
    ARs = [0.5,    0.5,   2.0,   5.0]
    Res = [1.0, 2000.0, 100.0, 100.0]
    #        x       o      o      o  Broken=x Working=o
    main(ARs, Res)
```

Listing 1: Code to create solutions

```
#### import the simple module from the paraview
from paraview.simple import *
import sys
#### disable automatic camera reset on 'Show'
paraview.simple._DisableFirstRenderCameraReset()

# get active source.
cavityClippedfoam = GetActiveSource()

# get active view
renderView1 = GetActiveViewOrCreate('RenderView')
# uncomment following to set a specific view size
# renderView1.ViewSize = [1054, 790]

# get color transfer function/color map for 'p'
pLUT = GetColorTransferFunction('p')
pLUT.RGBPoints = [0.0, 0.231373, 0.298039, 0.752941, 5e-17, 0.865003, 0.865003, 0.865003, 1e-16, 0.705882, 0.0156863,
pLUT.ScalarRangeInitialized = 1.0

```

```
# show data in view
cavityClippedfoamDisplay = Show(cavityClippedfoam, renderView1)
# trace defaults for the display properties.
cavityClippedfoamDisplay.ColorArrayName = ['POINTS', 'p']
cavityClippedfoamDisplay.LookupTable = pLUT
cavityClippedfoamDisplay.ScalarOpacityUnitDistance = 1.0844426982393176
cavityClippedfoamDisplay.SelectInputVectors = ['POINTS', 'U']
cavityClippedfoamDisplay.WriteLog = ''

# reset view to fit data
renderView1.ResetCamera()

# show color bar/color legend
cavityClippedfoamDisplay.SetScalarBarVisibility(renderView1, True)

# get opacity transfer function/opacity map for 'p'
pPWF = GetOpacityTransferFunction('p')
pPWF.Points = [0.0, 0.0, 0.5, 0.0, 1e-16, 1.0, 0.5, 0.0]
pPWF.ScalarRangeInitialized = 1

# change representation type
cavityClippedfoamDisplay.SetRepresentationType('Surface LIC')

# set scalar coloring
ColorBy(cavityClippedfoamDisplay, ('POINTS', 'streamFunction'))

# rescale color and/or opacity maps used to include current data range
cavityClippedfoamDisplay.RescaleTransferFunctionToDataRange(True)

# show color bar/color legend
cavityClippedfoamDisplay.SetScalarBarVisibility(renderView1, True)

# get color transfer function/color map for 'streamFunction'
streamFunctionLUT = GetColorTransferFunction('streamFunction')
streamFunctionLUT.RGBPoints = [-2.037569999694824, 0.231373, 0.298039, 0.752941, -0.9894677493721247, 0.865003, 0.865
streamFunctionLUT.ScalarRangeInitialized = 1.0

# get opacity transfer function/opacity map for 'streamFunction'
streamFunctionPWF = GetOpacityTransferFunction('streamFunction')
streamFunctionPWF.Points = [-2.037569999694824, 0.0, 0.5, 0.0, 0.058634500950574875, 1.0, 0.5, 0.0]
streamFunctionPWF.ScalarRangeInitialized = 1

# Properties modified on renderView1
renderView1.Background = [1.0, 1.0, 1.0]

# Properties modified on renderView1
renderView1.OrientationAxesVisibility = 0

# Properties modified on cavityClippedfoamDisplay
cavityClippedfoamDisplay.ColorMode = 'Multiply'

# hide color bar/color legend
cavityClippedfoamDisplay.SetScalarBarVisibility(renderView1, False)

# Properties modified on cavityClippedfoamDisplay
cavityClippedfoamDisplay.EnhanceContrast = 'LIC and Color'

#change interaction mode for render view
renderView1.InteractionMode = '2D'

#### saving camera placements for all active views

# current camera placement for renderView1
renderView1.InteractionMode = '2D'
renderView1.CameraPosition = [-6.546626850823488e-06, 0.024711792637900302, 0.29411509449255]
renderView1.CameraFocalPoint = [-6.546626850823488e-06, 0.024711792637900302, 0.004999999888241291]
renderView1.CameraParallelScale = 0.0768830580193085

#### uncomment the following to render all views
```

```
SaveScreenshot('/Users/localmin/code/MAE-219/cavityClipped/test_image.png', magnification=1, quality=100, view=render
sys.exit()
```

Listing 2: Code to generate pretty plots using paraFoam