Date: 15/Oct/2024

Technical information on the tool for automated CFD simulation of air flow inside the nasal cavity

Uday Tummala, Dr. Carlos Lange*
*email = clange@ualberta.ca (if any queries please contact this email)

Background: Obstructive sleep apnea (OSA) affects 2-4% of children and 9-38% of adults. OSA has a multifactorial etiology. Untreated OSA causes poor growth, behavioral, and cardiovascular problems. Multidisciplinary management is required to treat these patients. Orthodontic interventions might help in growing patients with a narrow upper jaw and a short lower jaw. Some of the orthodontic interventions include palatal expansion and lower jaw advancement. These treatments involve enlargement of the upper jaw, thus might alter the upper airway dimension. Understanding the possible air resistance changes in the upper airway after orthodontic procedures is important. Since cone beam computed tomography is used in some cases for orthodontic treatment planning and craniofacial discrepancy diagnosis, we aim to utilize this imaging exam to analyze the airflow.

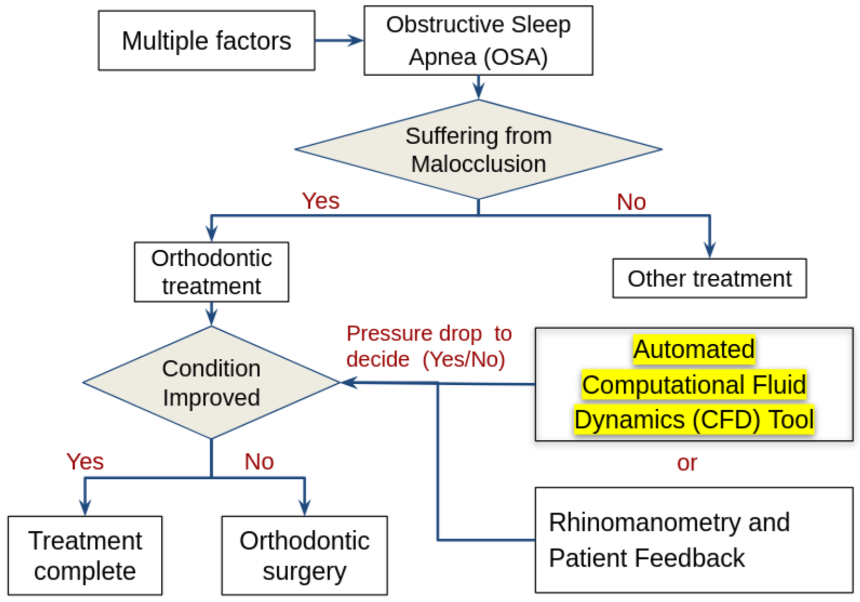

Multiple factors
Obstructive Sleep Apnea (OSA)
Suffering from Malocclusion
Yes
No
Orthodontic treatment
Other treatment
Pressure drop to decide (Yes/No)
Automated Computational Fluid Dynamics (CFD) Tool
or
Rhinomanometry and Patient Feedback
Condition Improved
Yes
No
Treatment complete
Orthodontic surgery

Fig.1: Treatment decision tree for usage of automated CFD tool

Objective: The main objective of this project is to create a tool that uses open source software to automate the CFD workflow so that it can be operated by health care technicians without any knowledge of CFD. The CFD is used to know the pressure drop and airway resistance of the nasal cavity of a patient suffering from obstructive sleep apnea.

Open source software's literature review: Open source software usually refers to computer software where users can use it to study, change, and distribute to anyone and for any purpose. Most of the open source software's, when compared to proprietary software's, lacks development in user friendliness, additional features, and in the case of CFD software's thoroughly tested and accurate numerical code. Hence, to choose the best open source software, its number of users, number of developers actively working on its development, features, accuracy in the case of CFD software, and its ability to automate the operations are compared to good proprietary software such as ANSYS for CFD and post-processing software, Autodesk Maya, and Meshmixer for pre-processing software. Based on this open source software's Blender, OpenFOAM and Paraview are chosen for pre-processing, CFD, and post-processing. For graphic user interface software, open source software like the Tkinter library from Python is used for its simplicity and for its need of fewer CPU memory requirements.

Methodology: In order for healthcare personnel to be able to operate the CFD software application without the knowledge of CFD, a user-friendly graphic user interface is created using the Python programming language and its libraries, Tkinter. This application needs a completed segmented geometry of the nasal cavity and pharynx, for which the CBCT is used to generate 3D images that are segmented automatically using an algorithm. The CFD software application consists of an algorithm created using the Python programming language that does preprocessing of geometry, modeling using CFD, and postprocessing of results automatically. It only uses open-source software such as Blender, OpenFOAM, and ParaView. At the end, the algorithm and the software application automatically give pressure and velocity images of the 3D-scanned model, along with the values of pressure drop and airway resistance.

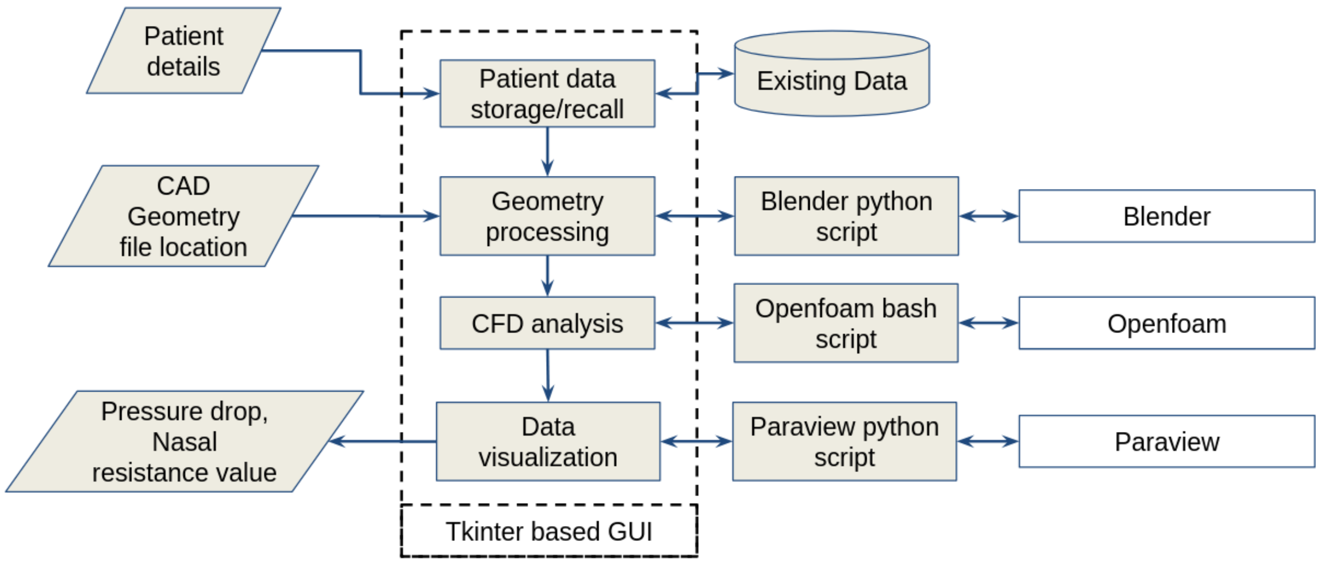


Fig.2: Visualization of operation of automated CFD workflow tool

Progress so far:

1. Completed preparation of scripts for Blender, Paraview, Tkinter and OpenFOAM softwares
2. Completed automation of CFD workflow and completed preliminary testing
3. Completed working prototype of automated CFD Tool (version 0.1)

Work need to be completed:

1. Complete final version of automated CFD Tool (version 1.0)
2. Automatic alignment of CAD model in Blender and exporting point location and area of Inlet face, outlet face
3. Update of paraview script file, K, Omega and transportProperties files similar to U file to include varying values based on input flow rate as shown in Fig.5.
4. Install OS OpenSUSE 15.6 in mini PC, respective software's and run the application
5. Testing of application with health care personnel and improving the graphic user interface of the application based on their feedback

Assumptions made in CFD simulation: Fluid flow inside the nasal cavity is steady, fluid is Air, Ideal gas, incompressible, outside atmospheric pressure is 91.5kPa and temperature is 22 degrees Celsius*.

* Keep this values and properties constant and if for any reason it needed change, change the values and commands in the corresponding files present inside *0*, *constant* and *system* folders present inside "*Master_CFD_file*" named folder.

Folder organization structure: The shared folder "Ortho_CFD_project" as shown in Fig. 3 consists of conference papers, documents, and geometry files from segmentation of the CBCT scan. It also consists of an application named "Ortho_App_0.1", it contains Blender, Openfoam, Paraview, and Tkinter script files. These files usually contain Python, C++, or bash commands. The application can be opened using the command shown in the below section "Steps to open application".

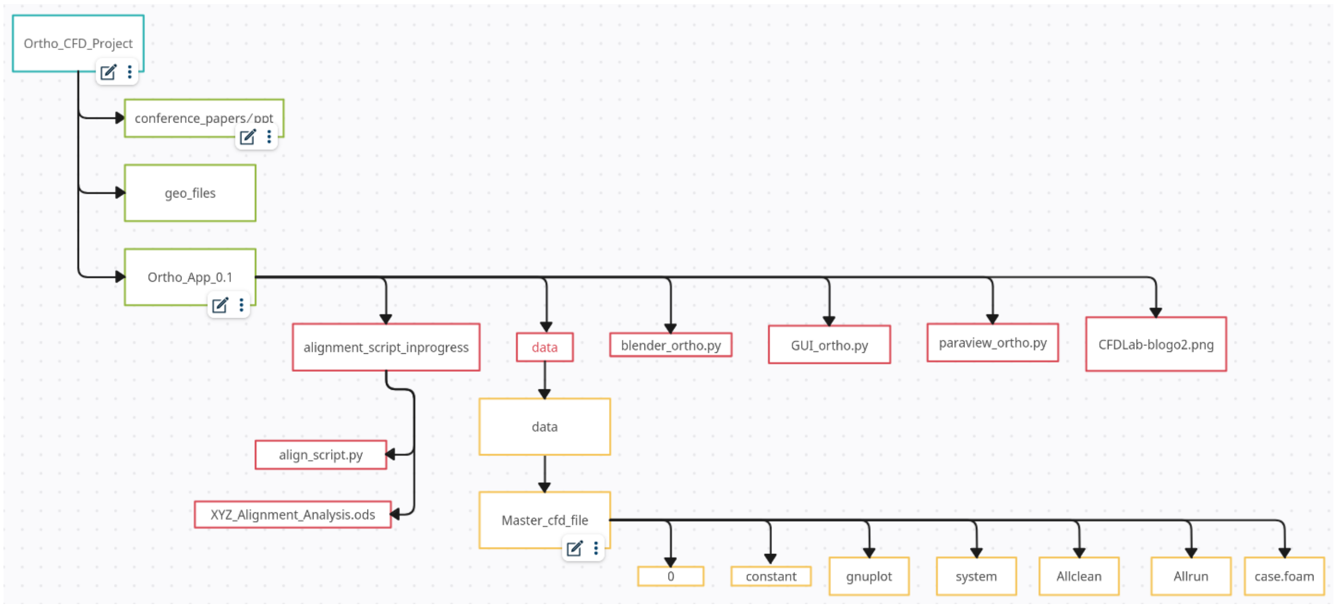
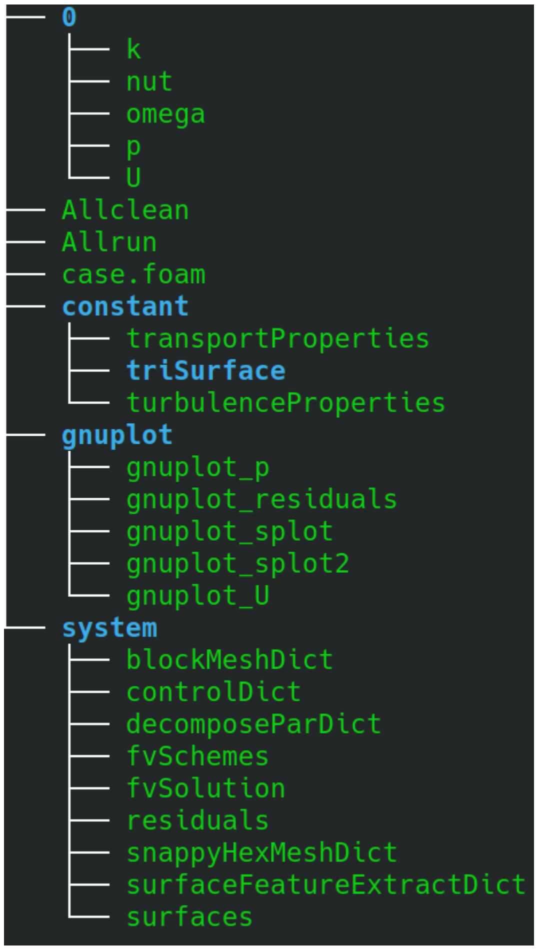


Fig.3: File organization structure of Ortho_CFD _project



```
├── 0
│   ├── k
│   ├── nut
│   ├── omega
│   ├── p
│   └── U
├── Allclean
├── Allrun
├── case.foam
├── constant
│   ├── transportProperties
│   ├── triSurface
│   └── turbulenceProperties
├── gnuplot
│   ├── gnuplot_p
│   ├── gnuplot_residuals
│   ├── gnuplot_splot
│   ├── gnuplot_splot2
│   └── gnuplot_U
├── system
│   ├── blockMeshDict
│   ├── controlDict
│   ├── decomposeParDict
│   ├── fvSchemes
│   ├── fvSolution
│   ├── residuals
│   ├── snappyHexMeshDict
│   ├── surfaceFeatureExtractDict
│   └── surfaces
```

Fig.4: Files inside Master_cfd_file

Requirements for operation of application:

1. Operating Software = OpenSUSE Leap 15.4 or 15.6
2. Required Software's:
   a). CFD Software = Openfoam v2306 or other*
   b). Pre-processing Software = Blender v2.82 or other*
   c). Post-processing software = Paraview v5.12 or other*
   d). GUI software = Tkinter Python 3.11 or other*

* If other change the location and version of software installed in its respective scripts

Software's installation procedure: Software's can be installed from Discover app store in OpenSUSE OS or through terminal by typing below commands.

a) OpenSUSE Leap 15.4 installation procedure:
   See website https://sites.ualberta.ca/~clange/linux.html for details

b) Openfoam installation commands:
   sudo zypper addrepo https://download.opensuse.org/repositories/science/15.6/science.repo
   sudo zypper refresh
   sudo zypper install openfoam2306-default

c) Python and Tkinter installation commands:
sudo zypper install python3-tk
sudo zypper remove python3-tk
sudo zypper install python311
sudo zypper install python311-tk

d) Blender and Paraview:
Install from Discover app store

Steps to open application:

1. Go into folder "*Ortho_App_0.1*"
2. Open Terminal by right clicking or by pressing Alt+Shift+F4
3. Type and execute command "*python3.11 GUI_ortho.py*"

Communication between softwares: See appendix for full Blender, OpenFOAM, Paraview and Tkinter scripts, this section focus on description on commands that are used to communicate between software's to aid in automation.

a) Blender gets the input file location from the file named "*geo_in.txt*" and exports the models into desired user location, this location is present in "*sdir.txt*", these txt files are created by Tkinter application. Blender also needs to export location of three points in the form of txt file.
b) OpenFOAM gets to know the user desired file location from "*sdir.txt*". Tkinter does copy/paste of the *Master_CFD_file* into a desired user location from "*sdir.txt*". Flow rate value is taken from "*pvfr.txt*" as shown in Fig.5. The txt file is created by Tkinter. OpenFOAM also requires location of three points inside the model one located near the inlet and other at the outlet, these values need to be stored in txt file and is called out in *SnappyHexMeshDict* and in *controlDict* file as shown below. Similarly Paraview uses the three points to know the location at which cross-section of pressure and velocity plot needed to taken and exported.
c) U, K and Omega values depends on the flow rate value, equations to find K and Omega can be found in any CFD text book or in internet by simply searching "CFD turbulence input equations". If flow rate is less than or equal to 15LPM the flow is laminar if it is greater than 15LPM then flow is turbulent and this needs to be updated in transportProperties file as shown below. RASModel can be kOmegaSST or kOmega.

transportProperties:

```
simulationType      laminar; // or RAS
RAS
{
   RASModel     kOmega; //kOmegaSST;

   turbulence      off;

  printCoeffs     on;
}
```

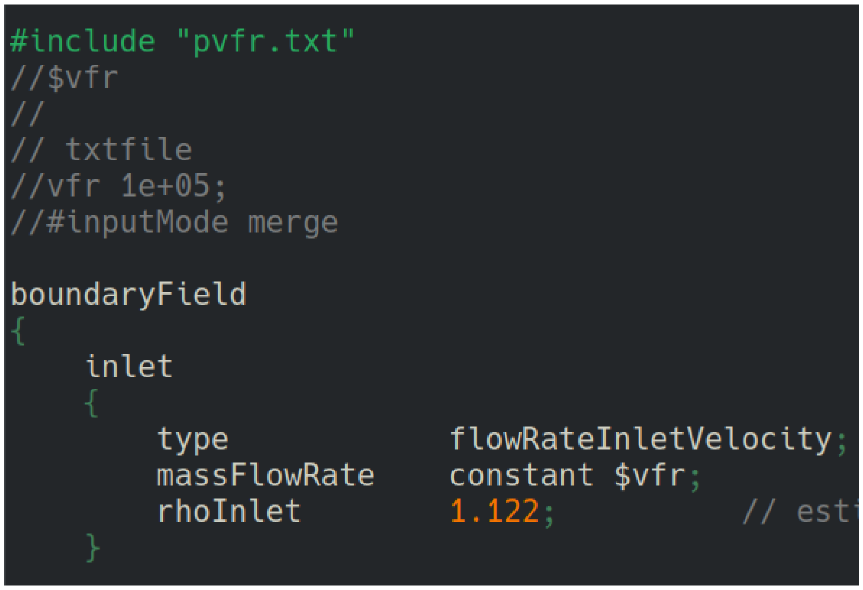
SnappyHexMeshDict:

```
locationInMesh (0.035 0.014 0.098); //in
```

controlDict:

```
sampledSurfaceDict
       {
         type      plane;
         point     (0.035 0.014 0.098); //in
         normal     (0 1 0);
       }

  probeLocations
  (
     (0.035 0.014 0.098) //in
     (0.045 0.014 0.098) //in
     (0.042 0.09 0.005) //out
  );
```



```
#include "pvfr.txt"
//$vfr
//
// txtfile
//vfr 1e+05;
//#inputMode merge

boundaryField
{
    inlet
    {
        type            flowRateInletVelocity;
        massFlowRate    constant $vfr;
        rhoInlet        1.122;          // est
    }
```

Fig.5: Script inside U file

d) Paraview exports the files into location based on "*sdir.txt*"

Automation of alignment of Model: Based on the analysis of 14 models, it is observed that some of the models are misaligned by 1–5 degrees around the y and z axis. The current method uses fixed value in the automation app to correct the models. It is derived from analyzing 14 models and determining the average and maximum misalignment value. Although in some models misalignment is not completely eliminated, the correction will be sufficient for the blender to create inlet and outlet faces. The inlet and outlet faces are created by Blender using boolean operations from boxes as shown in Fig. 8. The dimensions of the boxes and their location are based on the analysis of bounds of 14 models.

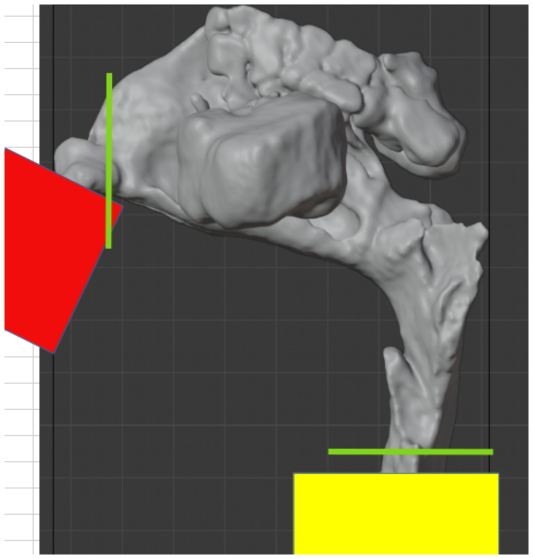


Fig.6 Inlet and outlet bounds of the model

In case of models with higher misalignment, the current algorithm will face difficulty in creating inlet and outlet faces for some models, hence an algorithm for misalignment shown below is developed. It is based on finding the optimum distance between two points present at the opposite ends of the bounding box of the model, this is done for xz and xy plane, it can be seen by violet line shown in below Fig.7. All the lines used to measure the distance passes through median point of the model, this median point can found out through inbuilt algorithm in Blender.

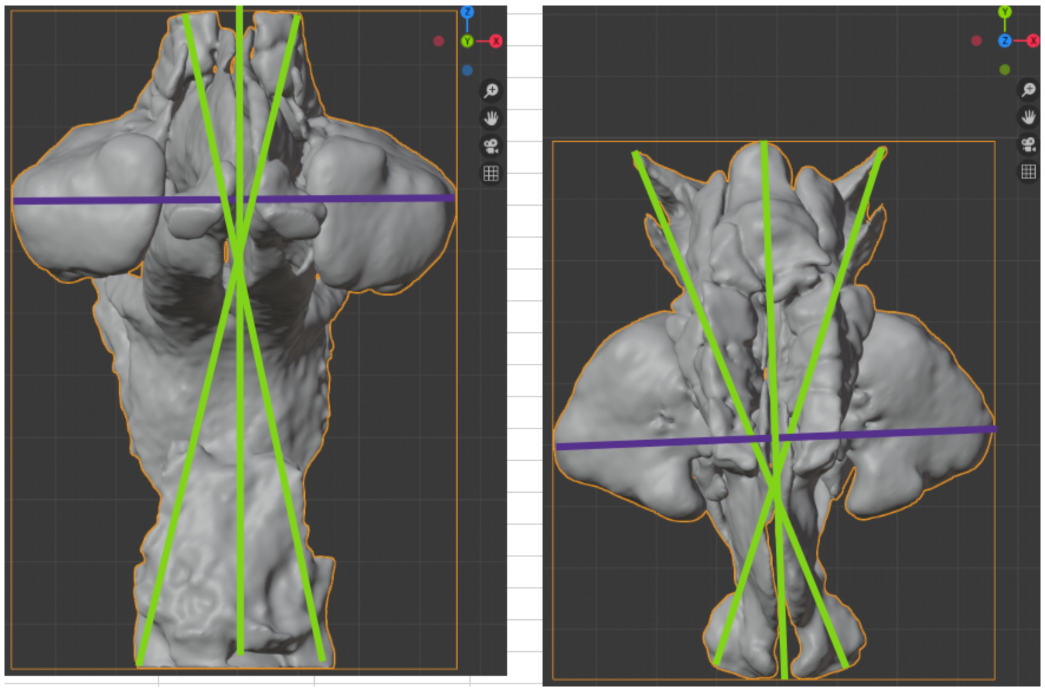


Fig.7: Minimum bounds line of model at various plane

Troubleshooting: If there is any problem with CFD results during testing of nasal cavity model, check whether the model from Blender software is fully closed or not by using below commands,
openfoam2306
surfaceCheck combined.stl >log.chk

- if log.chk shows model is open check Blender script file and segmented model from Meshmixer
- if it shows model is closed remove unwanted upper and lower sinuses from the 3D model in the Meshmixer or in Blender software and automate the process by adding appropriate commands inside Blender or Meshmixer script.

Conclusion: Automation of CFD workflow is achieved using open source software. The current tool can automatically process the segmented CBCT model and gives out values of pressure drop, airway resistance, and pictures of the contour plot. The current tool can automatically correct the model misalignment not greater than 10 degrees, if the model has misalignment of 10 or more degrees, the current code needs further development for the tool to be able to correct the misalignment.

Appendix:

Blender python script:

```
# Msc Ortho CFD Blender 2.82 script File
# Computational Fluid Dynamics Lab
# Author = Uday Tummala
# Dr. Carlos F. Lange
# Department of Mechanical Engineering
# University of Alberta
#
# Date 2023-12-27
# cmd = blender --background --python blender_ortho.py

import bpy  # Blender Python API
import math
import mathutils
import time
import os

start_time = time.time()  #start time to see how much time blender takes to preprocess the geometry file

##########################################################################################
# delete all existing objects
bpy.ops.object.mode_set(mode='OBJECT')
bpy.ops.object.select_all(action='SELECT')
bpy.ops.object.delete()

##########################################################################################
# import stl file
file0 = open("sdir.txt", "r")
path0 = file0.readline()
file0.close()
path00 = os.path.join(path0, "geo_in.txt")
file = open(path00, "r")
path1 = file.readline() #assign path value from
file.close()
bpy.ops.import_mesh.stl(filepath= path1)

##########################################################################################
# Move model to origin
bpy.ops.object.origin_set(type='GEOMETRY_ORIGIN', center='MEDIAN')
active_obj = bpy.context.active_object
active_object_verts = active_obj.data.vertices
xValues = []
yValues = []
zValues = []
```

```
#########################################################################################
# Alignment in y, z axis
# In progress, see align_script_inprogress for details
bpy.context.object.rotation_euler = (0,0,-0.05) # -0.01-p4t1,0.02-p4t2, -0.065-p5t1, 0.05-p5_val, 0.044

#########################################################################################
# Removing floating objects/artifacts
minx =  min(xValues)    # find minimum value from xValues string
maxx =  max(xValues)  # find maximum value from xValues string
miny =  min(yValues)
maxy =  max(yValues)
minz =  min(zValues)
maxz =  max(zValues)

#########################################################################################
# Move to +ve co-ordinates
bpy.context.object.location[0] = abs(minx)
bpy.context.object.location[1] = abs(miny)
bpy.context.object.location[2] = abs(minz)
bpy.ops.object.mode_set(mode='EDIT')
bpy.ops.mesh.select_all(action='SELECT')
bpy.ops.mesh.bisect(plane_co=(0, 0, 2), plane_no=(0, 0, 1), use_fill=True, clear_inner=False,
xstart=1000, xend=0, ystart=0, yend=0)  # use bisect at location and direction
bpy.ops.mesh.loop_to_region(select_bigger=True)
bpy.ops.mesh.separate(type='SELECTED')
bpy.ops.object.mode_set(mode='OBJECT')
bpy.ops.object.select_all(action='DESELECT')
model = bpy.context.active_object
bpy.context.active_object.select_set(True)
bpy.ops.object.delete()  # delete selected body

#########################################################################################
# Make Outlet face
bpy.ops.object.select_all(action='SELECT')
for obj in bpy.context.selected_objects:
    obj.name = "wall"
    obj.data.name = "wall"

bpy.ops.object.select_all(action='DESELECT')
bpy.ops.mesh.primitive_cube_add(size=2, enter_editmode=False, location=(0, 0, 0))  #add cube of
size 2
cube = bpy.context.active_object   # name current model as cube
bpy.context.view_layer.objects.active = cube
bpy.context.object.scale[0] = 100
bpy.context.object.scale[1] = 100
bpy.context.object.scale[2] = 10
ob = bpy.data.objects["wall"]
ob.select_set( state = True, view_layer = bpy.context.view_layer )
bpy.context.view_layer.objects.active = ob      # name current selection as ob
bpy.ops.object.modifier_add(type='BOOLEAN')    # perform boolean operation
bpy.context.object.modifiers["Boolean"].operation = 'DIFFERENCE'
bpy.context.object.modifiers["Boolean"].object = cube
bpy.ops.object.modifier_apply({"object": ob}, modifier="Boolean")
bpy.ops.object.select_all(action='DESELECT')
bpy.context.view_layer.objects.active = cube
bpy.context.active_object.select_set(True)
bpy.ops.object.delete()  # delete selected object
bpy.context.view_layer.objects.active = ob
bpy.ops.object.mode_set(mode='EDIT')
bpy.ops.mesh.select_mode(type="FACE")
bpy.ops.mesh.separate(type='SELECTED')
for obj in bpy.context.selected_objects:
    obj.name = "outlet"
    obj.data.name = "outlet"


bpy.ops.object.mode_set(mode='OBJECT')
out = bpy.context.active_object
bpy.ops.object.select_all(action='DESELECT')

##########################################################################################
# Make Inlet face
bpy.ops.mesh.primitive_cube_add(size=2, enter_editmode=False, location=(0, 0, 0))
cube= bpy.context.active_object
bpy.context.view_layer.objects.active = cube
bpy.context.object.scale[0] = 100
bpy.context.object.scale[1] = 10
bpy.context.object.scale[2] = 10
bpy.context.object.rotation_euler[0] = 1
bpy.context.object.location[2] = 62
bpy.context.view_layer.objects.active = ob
bpy.ops.object.modifier_add(type='BOOLEAN')
bpy.context.object.modifiers["Boolean"].operation = 'DIFFERENCE'
bpy.context.object.modifiers["Boolean"].object = cube
bpy.ops.object.modifier_apply({"object": ob}, modifier="Boolean")
bpy.ops.object.select_all(action='DESELECT')
bpy.context.view_layer.objects.active = cube
bpy.context.active_object.select_set(True)
bpy.ops.object.delete()
bpy.context.view_layer.objects.active = ob
bpy.ops.object.mode_set(mode='EDIT')
bpy.ops.mesh.select_mode(type="FACE")
bpy.ops.mesh.separate(type='SELECTED')
for obj in bpy.context.selected_objects:
    obj.name = "inlet"
    obj.data.name = "inlet"
```

```
bpy.ops.object.mode_set(mode='OBJECT')
inl = bpy.context.active_object
bpy.ops.object.select_all(action='DESELECT')

#####################################################################################
# Export as STL
path2 = os.path.join(path0, "constant/triSurface/")
bpy.ops.export_mesh.stl(filepath=path2, check_existing=True, filter_glob='*.stl', use_selection=False,
global_scale=1.0, use_scene_unit=False, ascii=True, use_mesh_modifiers=True,
batch_mode='OBJECT', axis_forward='Y', axis_up='Z')

#####################################################################################
# Export IMAGE
# create color
mat = bpy.data.materials.new('Material1')
mat.diffuse_color = (0.8, 0.00269661, 0.00091005, 1)
mat1 = bpy.data.materials.new('Material2')
mat1.diffuse_color = (0.0103095, 0.8, 0.0170713, 1)

#####################################################################################
# get the object
obj = bpy.data.objects['wall']

#####################################################################################
# get the material
mat = bpy.data.materials['Material1']  # name Material1 as mat
mat.use_nodes = True
bpy.data.worlds["World"].node_tree.nodes["Background"].inputs[1].default_value = 5.3

#####################################################################################
# assign material to object
obj.data.materials.append(mat)  # assign mat as material to our model obj
bpy.ops.object.select_all(action='DESELECT')
obj1 = bpy.data.objects['inlet']
mat1 = bpy.data.materials['Material2']
obj1.data.materials.append(mat1)
mat1.use_nodes = True
bpy.data.worlds["World"].node_tree.nodes["Background"].inputs[1].default_value = 5.3
bpy.ops.object.select_all(action='DESELECT')
obj2 = bpy.data.objects['outlet']
mat1 = bpy.data.materials['Material2']
mat1.use_nodes = True
obj2.data.materials.append(mat1)
bpy.data.worlds["World"].node_tree.nodes["Background"].inputs[1].default_value = 5.3
bpy.ops.object.select_all(action='DESELECT')

#####################################################################################
# create light
light_data = bpy.data.lights.new(name="light-data1", type='POINT')
```

```
light_data.energy = 5000

########################################################################################
# Create new object, pass the light data
light1 = bpy.data.objects.new(name="light1", object_data=light_data)

########################################################################################
# Link object to collection in context
bpy.context.collection.objects.link(light1)
light1.location = (abs(minx)+abs(maxx), abs(miny)+abs(maxy), abs(minz)+abs(maxz))

########################################################################################
# create the camera
scn = bpy.context.scene
cam_ob1 = bpy.data.cameras.new("camera1")
cam_ob1.lens = 45
cam1 = bpy.data.objects.new("camera1", cam_ob1)
cam1.location = ((abs(minx)+abs(maxx))*2.15, -(abs(minz)+abs(maxz))/2, -(abs(miny)+abs(maxy))/2)
cam1.rotation_euler = (math.radians(0), math.radians(125), math.radians(320))
scn.collection.objects.link(cam1)
bpy.context.scene.camera = bpy.data.objects['camera1']
bpy.context.scene.cycles.samples = 1
scn.render.use_border = True
bpy.context.scene.render.resolution_x = 600
bpy.context.scene.render.resolution_y = 600
bpy.context.scene.render.resolution_percentage = 100
bpy.context.scene.render.image_settings.file_format='JPEG'
path3 = os.path.join(path0,"blendout.jpg")
bpy.context.scene.render.filepath = path3
bpy.context.scene.view_settings.exposure = 2.25
bpy.ops.render.render('INVOKE_DEFAULT', write_still=True)

########################################################################################
# Quit Blender
print("Finished running preprocessing of model in Blenderv2.82")
print("Please check output image blendout.jpg")
print("Time taken for preprocessing: ",time.time()-start_time)
bpy.ops.wm.quit_blender()
```

Misalignment Blender script:

```
def my_function():
   for v in active_object_verts:
      if v.select == True:
         bpy.ops.object.transform_apply(location=True, rotation=True, scale=True)
         xValues.append(v.co[0])
         yValues.append(v.co[1])
         zValues.append(v.co[2])

for z in range(-4, 4, 1):  #  iteration of z in range 0f +/- 4 degree with an increment of 1 deg
   for y in range(-4, 4, 1):
      xValues.clear()
      yValues.clear()
      zValues.clear()
      bpy.context.object.rotation_euler = (0,math.radians(y),math.radians(z))
      my_function()
      minx =  min(xValues)
      maxx =  max(xValues)
      x1 = xValues.index(minx)
      x2 = xValues.index(maxx)
      miny = yValues[x1]
      maxy = yValues[x2]
      minz = zValues[x1]
      maxz = zValues[x2]
      d1.append(pow((pow((maxx-minx), 2) + pow((maxy-miny), 2) + pow((maxz-minz), 2)), 0.5))
      angy1.append(y)
      angz1.append(z)

mind1 = min(d1)
x3 = d1.index(mind1)
bpy.context.object.rotation_euler = (0,math.radians(angy1[x3]),math.radians(angz1[x3]))
```

Tkinter GUI script:

```
# Msc Ortho CFD python tkinter script File
# Computational Fluid Dynamics Lab
# Author = Uday Tummala
# Dr. Carlos F. Lange
# Department of Mechanical Engineering
# University of Alberta
#
# Date 2024-01-07
# cmd = python3.11 GUI_ortho.py

from tkinter import *
import os
import openpyxl
import subprocess
import shutil
import pandas as pd
import csv
from PIL import ImageTk,Image
from tkinter import messagebox
from tkinter import filedialog
from tkinter.filedialog import askdirectory
from tkinter import ttk

# Window Title, size
root = Tk()
root.title("Ortho CFD v0.1")
photo = ImageTk.PhotoImage(Image.open('CFDLab-blogo2.png'))
root.wm_iconphoto(False, photo)
root.geometry("600x800")
root.minsize(height=600, width=800)
root.maxsize(height=700, width=1000)

#####################################################################################
# Tab 1, Welcome page
def tab1():
    label1=Label(root, text='\n Univeristy of Alberta\n Ortho CFD Applicaton version 0.1\n\nWelcome', 
font=('Times_New_Roman', 25), bg = "green", bd = 50, fg = "white")
    label1.pack()
    label2=Label(root, text='\nThis App helps determine the pressure difference in the \nNasal cavity 
from 3D X-RAY Scan. This is done by using\nCFD analysis, the input file format must be in stl or obj.\n
This App uses blender, openfoam and Paraview softwares.\n\n\nThis App is developed by University of 
Alberta.\nFor more information please contact Dr Carlos Lange\n email = clange@ualberta.ca', 
font=('Times_New_Roman', 15))  # define label2 properties
    label2.pack()
    button1=Button(root, text='Next', font=('Times_New_Roman',16), command=next_win1, 
activebackground='blue')  #define button properties
    button1.pack(side=BOTTOM)   # place button1 at bottom
```

```
################################################################################
# Tab 2, asks for information on patient and file details
def tab2():
    global un   # define global variable un
    global pn   # define global variable pn
    global pa
    global pd
    global fn
    un = StringVar()  # define un as string variable
    pn = StringVar()
    pa = StringVar()
    pd = StringVar()
    fn = StringVar()
    un.set("")   # define un as empty matrix
    pn.set("")
    pa.set("")
    pd.set("")
    fn.set("")
    user_name = Label(root, text = "Username",font=('Times_New_Roman', 16)).place(x = 40, y = 60)
    patient_name = Label(root, text = "Patient Name",font=('Times_New_Roman', 16)).place(x = 40, y
= 120)
    patient_Age = Label(root, text = "Patient Age",font=('Times_New_Roman', 16)).place(x = 40, y =
180)
    patient_Doctor = Label(root, text = "Doctor Name",font=('Times_New_Roman', 16)).place(x = 40,
y = 240)
    user_name_e = Entry(root, width = 30, textvariable = un, font=('Times_New_Roman', 16)).place(x
= 300, y = 60)
    patient_name_e = Entry(root, width = 30, textvariable = pn, font=('Times_New_Roman',
16)).place(x = 300,y = 120)
    patient_age_e = Spinbox(root,from_=1, to=120, textvariable = pa, width =
8,font=('Times_New_Roman', 16)).place(x = 300, y = 180)
    patient_doctor_e = Entry(root, width = 30, textvariable = pd, font=('Times_New_Roman',
16)).place(x = 300, y = 240)
    Label(root, text = "Type new or previous folder\nname to save files",font=('Times_New_Roman',
16)).place(x = 40, y = 330)
    fn_entry = Entry(root, width = 20, textvariable = fn, font=('Times_New_Roman', 16)).place(x =
400, y = 330)  # define entry box here

    def open1():
        filepath = "gui_data.xlsx"
        if os.path.exists(filepath):
            top = Toplevel()
            top.title('Excel viewer')
            top.geometry("1200x400")
            top.minsize(height=400, width=1200)
            photo = ImageTk.PhotoImage(Image.open('CFDLab-blogo2.png'))
            top.wm_iconphoto(False, photo)
            wb1 = openpyxl.load_workbook(filepath)
            ws = wb1['tab1']
            max_row = ws.max_row
            max_col = ws.max_column
            global se
            se = IntVar()
            s_e = Spinbox(top,from_=1, to=max_row-1, width=6, textvariable = se,
font=('Times_New_Roman', 14))
            s_e.pack()
            lb = Label(top, text = ("Which data to display of total", max_row-1),
font=('Times_New_Roman', 14))
            lb.pack()

            def sh_fun():
                dd.insert(INSERT, "\n")
                for i in range(1, max_col):
                    if ws.cell(row = se.get()+1, column = i):
                        c_ob = ws.cell(row = se.get()+1, column = i)
                        dd.insert(END, ("--",c_ob.value))

            sb1 = Button(top, text='Show', font=('Times_New_Roman',14), command = sh_fun,
activebackground='blue')
            sb1.pack()
            dd = Text(top, height = 20, width = 130, font=('Times_New_Roman', 12), bg = "light yellow",
wrap=WORD)
            for i in range(1, max_col):
                row_text = ws.cell(row = 1, column = i)
                dd.insert(INSERT, ("--",row_text.value))
                dd.tag_add("start", "1.00","1.9999")
                dd.tag_config("start", background= "black", foreground= "white")

            dd.pack()
        else:
            messagebox.showwarning("Error", "No Data file")

    Label(root, text = "Click here to see previous\nfolder data",font=('Times_New_Roman', 16)).place(x
= 40, y = 420)
    open_button = Button(root, text = "Open", command=open1, font=('Times_New_Roman', 16),
activebackground='green', relief=GROOVE).place(x = 400, y = 420)

    for i in range(3):
        root.columnconfigure(i, weight=1)

    root.rowconfigure(1, weight=1)
    next_button = Button(root, text = "Next", font=('Times_New_Roman', 16), command =next_win2,
activebackground='blue').grid(row=2, column=2, sticky='e')
    back_button = Button(root, text = "Back", font=('Times_New_Roman', 16), command =back_win2,
activebackground='blue').grid(row=2, column=0, sticky='w')
```

```
##########################################################################################
# Tab 3, asks for Geometry location, and runs blender script and shows image
def tab3():
    global spp
    spp = StringVar()
    spp.set("")
    global file_path_var
    file_path_var = StringVar()
    file_path_var.set("")
    loc_label = Label(root, text = "Please open 3D-model input file",font=('Times_New_Roman',
16)).place(x = 40, y = 60)

    def open1():
        root_filename = filedialog.askopenfilename(initialdir="", title="select a file", filetypes=(("stl
files", "*.stl"),("obj files", "*.obj")))
        loc_Label = Label(root, text=root_filename, font=('Times_New_Roman', 16),fg="white",
bg="blue").place(x = 40, y = 100)
        file_path_var.set(root_filename)
        file_path_var.get()
        path1 = os.path.join(dirs, "geo_in.txt")
        file1 = open(path1,"w")
        file1.write(root_filename)
        file1.close()

    def run1():
        status_e.delete(0, END)
        status_e.insert(0, "")
        source_dir = r"data/Master_cfd_file"
        shutil.copytree(source_dir, dirs, symlinks=False, ignore=None, ignore_dangling_symlinks=False,
dirs_exist_ok=True)
        proc = subprocess.Popen(["blender --background --python blender_ortho.py"], stdout =
subprocess.PIPE, shell=True)
        (out, err) = proc.communicate()
        if b"Finished" in out:
            status_e.insert("insert","completed")
        else:
            status_e.insert("insert","error")

    open_button = Button(root, text="Open", command=open1, activebackground='green',
            font=('Times_New_Roman', 16)).place(x = 400, y = 60)
    Run_Pre = Label(root, text = "Run Pre-Processing of Geometry",font=('Times_New_Roman',
16)).place(x = 40, y = 160)
    Run_button = Button(root, text = "Run", command=run1,font=('Times_New_Roman', 16),
activebackground='green', relief=GROOVE).place(x = 400, y = 160)
    status = Label(root, text = "Status of Geometry Pre-Processing", font=('Times_New_Roman',
16)).place(x = 40, y = 240)
    progressbar = ttk.Progressbar(mode="indeterminate")
    progressbar.start()
    progressbar.place(x = 40, y = 280, height=28, width=440)
    progressbar.step(25)
    status_e = Entry(root, width = 30, borderwidth=4, textvariable = spp, font=('Times_New_Roman',
14),bg="blue", fg="white")
    status_e.place(x = 40, y = 320)
    Click = Label(root, text = "Click here to see geometry", font=('Times_New_Roman', 16)).place(x =
40, y = 400)

    def open2():
        top = Toplevel()
        top.title('Geo IMG after Pre-proc')
        top.wm_iconphoto(False, photo)
        path = os.path.join(dirs, "blendout.jpg")
        my_img = ImageTk.PhotoImage(Image.open(path))
        panel = Label(top, image=my_img)
        panel.pack()
        panel.photo = my_img

    click_button = Button(root, text = "Open", font=('Times_New_Roman', 16),
activebackground='green',command=open2,relief=GROOVE).place(x = 400, y = 400)
    global cvar
    cvar = IntVar()
    c = Checkbutton(root, text="Check this box if displayed image of geometry good",
font=('Times_New_Roman', 16), variable=cvar, onvalue=1, offvalue=0)
    c.deselect()
    c.place(x = 40, y = 480)

    if os.path.exists(os.path.join(dirs, "geo_in.txt")) and os.path.exists(os.path.join(dirs,
"blendout.jpg")):
        status_e.delete(0, END)
        status_e.insert(0, "completed")
        path101 = os.path.join(dirs, "geo_in.txt")
        file99 = open(path101, "r")
        vv1 = file99.readline()
        file99.close()
        loc_Label = Label(root, text=vv1, font=('Times_New_Roman', 16),fg="white",
bg="blue").place(x = 40, y = 100)

    for i in range(3):
        root.columnconfigure(i, weight=1)

    root.rowconfigure(1, weight=1)
    next_button = Button(root, text = "Next", font=('Times_New_Roman', 16), command =next_win3,
activebackground='blue').grid(row=2, column=2, sticky='e')
    back_button = Button(root, text = "Back", font=('Times_New_Roman', 16), command =back_win3,
activebackground='blue').grid(row=2, column=0, sticky='w')

##########################################################################################
# Tab 4, runs openfoam and shows residuals progress
def tab4():
    global pvfr
    global scfd
    pvfr = StringVar()
    scfd = StringVar()
    pvfr.set("")
    scfd.set("")
    patient_VFR = Label(root, text = "Patient VFR\n (Volume Flow rate,
LPM)",font=('Times_New_Roman', 16)).place(x = 40, y = 60)
    patient_VFR_e = Spinbox(root,from_=1, to=500, textvariable = pvfr, font=('Times_New_Roman',
16)).place(x = 300, y = 60)
    ##
    def run1():
        status_e.delete(0, END)
        status_e.insert(0, "")
        path0 = os.path.join(dirs, "0", "pvfr.txt")
        file1 = open(path0,"w")
        file1.write("vfr" + " " + pvfr.get() + ";" + "\n#inputMode merge")
        file1.close()
        path2 = os.path.join(dirs, "constant/triSurface")
        proc1 = subprocess.Popen(['/bin/bash', '-c',"bash ./Allclean"], cwd=dirs,
stdout=subprocess.PIPE)
        proc1.wait()
        procc = subprocess.Popen(['/bin/bash', '-c',"rm combined.stl"], cwd=path2,
stdout=subprocess.PIPE)
        procc.wait()
        proc2 = subprocess.Popen(["cat *.stl >> combined.stl"], cwd=path2, shell=True,
stdout=subprocess.PIPE)
        proc2.wait()
        proc = subprocess.Popen(['/bin/bash', '-c',"bash ./Allrun"], cwd=dirs, stdout=subprocess.PIPE)
        (out, err) = proc.communicate()
        if b"Finished" in out:
            status_e.insert("insert","completed")
        else:
            status_e.insert("insert","error")

    Run_CFD = Label(root, text = "Run CFD of the Geometry",font=('Times_New_Roman',
16)).place(x = 40, y = 160)
    Run_button = Button(root, text = "Run", command=run1, font=('Times_New_Roman', 16),
activebackground='green', relief=GROOVE).place(x = 400, y = 160)
    Click = Label(root, text = "Click here to see CFD progress \n graph",font=('Times_New_Roman',
16)).place(x = 40, y = 260)

    def open1():
        top = Toplevel()
        top.title('Image of CFD progress graph')
        def refresh1():
            path2 = os.path.join(dirs, "gnuplot")
            proc1 = subprocess.Popen(['/bin/bash', '-c',"bash ./gnuplot_residuals"], cwd=path2,
stdout=subprocess.PIPE)
```

```
            proc1.wait()

        path2 = os.path.join(dirs, "gnuplot", "residual_plot.png")
        if os.path.exists(path2):
            top.wm_iconphoto(False, photo)
            my_img1 = ImageTk.PhotoImage(Image.open(path2))
            img_label = Label(top, image=my_img1)
            img_label.pack()
            img_label.photo = my_img1
        else:
            refresh1

        btn1 = Button(top, text="Refresh", command = refresh1, activebackground='green',
font=('Times_New_Roman', 16)).pack()
    ##
    click_button = Button(root, text = "Open", font=('Times_New_Roman', 16),
activebackground='green',command=open1,relief=GROOVE).place(x = 400, y = 260)
    status = Label(root, text = "Status of CFD Analysis",font=('Times_New_Roman', 16)).place(x = 40,
y = 340)
    progressbar = ttk.Progressbar(mode="indeterminate")
    progressbar.start()
    progressbar.place(x = 40, y = 380, height=28, width=400)
    progressbar.step(25)
    status_e = Entry(root, width = 30, borderwidth=4, textvariable = scfd, font=('Times_New_Roman',
14),bg="blue", fg="white", )
    status_e.place(x = 40, y = 420)
    if os.path.exists(os.path.join(dirs, "260", "U")) and os.path.exists(os.path.join(dirs, "0", "pvfr.txt")):
            status_e.delete(0, END)
            status_e.insert(0, "completed")

    for i in range(3):
        root.columnconfigure(i, weight=1)

    root.rowconfigure(1, weight=1)
    next_button = Button(root, text = "Next", font=('Times_New_Roman', 16), command =next_win4,
activebackground='blue').grid(row=2, column=2, sticky='e')
    back_button = Button(root, text = "Back", font=('Times_New_Roman', 16), command =back_win4,
activebackground='blue').grid(row=2, column=0, sticky='w')

##########################################################################################
# Tab 5, runs paraview script and shows pressure, velocity plots and displays values of pressure drop
and airway resistance
def tab5():
    def run1():
        status_e.delete(0, END)
        status_e.insert(0, "")
        path2 = os.path.join(dirs, "postProcessing/avgsurf_in/0/surfaceFieldValue.dat")
        path3 = os.path.join(dirs, "postProcessing/avgsurf_out/0/surfaceFieldValue.dat")
        with open(path2) as f:
            reader = csv.reader(f, delimiter="\t")
            for line in reader:
                print(line)

        p_in = float(line[1])

        with open(path3) as f:
            reader = csv.reader(f, delimiter="\t")
            for line in reader:
                print(line)

        p_out = float(line[1])
        p_drop_e.insert("insert",(p_in - p_out) / 1000)
        a_res_e.insert("insert",(p_in - p_out) / (1000 * float(pvfr.get())))
        proc = subprocess.Popen(["pvbatch paraview_ortho.py"], stdout=subprocess.PIPE, shell=True)
        (out, err) = proc.communicate()
        if b"Finished" in out:
            status_e.insert("insert","completed")
        else:
            status_e.insert("insert","error")

    Run_post = Label(root, text = "Run Post-Processing of the CFD",font=('Times_New_Roman',
16)).place(x = 40, y = 60)
    Run_button = Button(root, text = "Run", command = run1, font=('Times_New_Roman', 16),
activebackground='green', relief=GROOVE).place(x = 400, y = 60)
    click_label = Label(root, text = "Click here to see pressure, velocity plots of the
Geometry",font=('Times_New_Roman', 16)).place(x = 40, y = 160)
    ###
    def open1():
        top = Toplevel()
        top.title('pressure, velocity plots')
        top.wm_iconphoto(False, photo)
        path1 = os.path.join(dirs, "p_cut_1.png")
        path2 = os.path.join(dirs, "p_cut_2.png")
        path3 = os.path.join(dirs, "p_full_1.png")
        path4 = os.path.join(dirs, "p_full_2.png")
        path5 = os.path.join(dirs, "p_full_3.png")
        path6 = os.path.join(dirs, "p_full_4.png")
        path7 = os.path.join(dirs, "v_cut_1.png")
        path8 = os.path.join(dirs, "v_cut_2.png")
        path9 = os.path.join(dirs, "v_streamlines1.png")
        path10 = os.path.join(dirs, "v_streamlines2.png")
        my_img1 = ImageTk.PhotoImage(Image.open(path1))
        my_img2 = ImageTk.PhotoImage(Image.open(path2))
        my_img3 = ImageTk.PhotoImage(Image.open(path3))
        my_img4 = ImageTk.PhotoImage(Image.open(path4))
        my_img5 = ImageTk.PhotoImage(Image.open(path5))
        my_img6 = ImageTk.PhotoImage(Image.open(path6))
        my_img7 = ImageTk.PhotoImage(Image.open(path7))
        my_img8 = ImageTk.PhotoImage(Image.open(path8))
        my_img9 = ImageTk.PhotoImage(Image.open(path9))
        my_img10 = ImageTk.PhotoImage(Image.open(path10))
        image_list = [my_img1, my_img2, my_img3, my_img4, my_img5, my_img6, my_img7, my_img8,
my_img9, my_img10]
        my_label = Label(top,image=my_img1)
        my_label.pack()
        ##
        def forward(image_number):
            global my_label
            global button_forward
            global button_back
            for widgets in top.winfo_children():
                widgets.destroy()

            my_label = Label(top,image=image_list[image_number-1])
            button_forward = Button(top, text=">>",font=('Times_New_Roman', 26), command=lambda:
forward(image_number+1))
            button_back = Button(top, text="<<", font=('Times_New_Roman', 26),command=lambda:
back(image_number-1))
            if image_number == 10:
                button_forward = Button(top, text=">>", state=DISABLED)

            my_label.pack()
            button_back.pack(side = LEFT)
            button_forward.pack(side = RIGHT)
        ##
        def back(image_number):
            global my_label
            global button_forward
            global button_back
            for widgets in top.winfo_children():
                widgets.destroy()

            my_label = Label(top,image=image_list[image_number-1])
            button_forward = Button(top, text=">>",font=('Times_New_Roman', 26), command=lambda:
forward(image_number+1))
            button_back = Button(top, text="<<",font=('Times_New_Roman', 26),command=lambda:
back(image_number-1))
            if image_number == 10:
                button_back = Button(top, text="<<",font=('Times_New_Roman', 26),state=DISABLED)

            my_label.pack()
            button_back.pack(side = LEFT)
            button_forward.pack(side = RIGHT)
        ##
        button_back = Button(top, text="<<",font=('Times_New_Roman', 26), command=back,
state=DISABLED)
```

```
        button_forward = Button(top, text=">>",font=('Times_New_Roman', 26), command=lambda:
forward(2))
        button_back.pack(side = LEFT)
        button_forward.pack(side = RIGHT)
    ##
    click_button = Button(root, text = "Open", font=('Times_New_Roman', 16),
activebackground='green', command=open1, relief=GROOVE).place(x = 600, y = 160)
    status = Label(root, text = "Status of Post-processing",font=('Times_New_Roman', 16)).place(x =
40, y = 240)
    progressbar = ttk.Progressbar(mode="indeterminate")
    progressbar.start()
    progressbar.place(x = 40, y = 280, height=28, width=400)
    progressbar.step(25)
    global spop
    global pdr
    global arr
    spop = StringVar()
    pdr = StringVar()
    arr = StringVar()
    spop.set("")
    pdr.set("")
    arr.set("")
    status_e = Entry(root, width = 30, borderwidth=4, textvariable = spop, font=('Times_New_Roman',
16),bg="blue", fg="white")
    status_e.place(x = 40, y = 320)
    p_drop = Label(root, text = "Pressure drop (kPa)",font=('Times_New_Roman', 16)).place(x = 40, y
= 400)
    p_drop_e = Entry(root, width = 15, textvariable = pdr, font=('Times_New_Roman', 16))
    p_drop_e.place(x = 300, y = 400)
    a_res = Label(root, text = "Air Resistance \n(cmH20/L/s)", font=('Times_New_Roman', 16)).place(x
= 40, y = 440)
    a_res_e = Entry(root, width = 15, textvariable = arr, font=('Times_New_Roman', 16))
    a_res_e.place(x = 300, y = 440)
    Label(root, text = "Save above data", font=('Times_New_Roman', 16)).place(x = 40, y = 510)
    ##
    if os.path.exists(os.path.join(dirs, "postProcessing/avgsurf_in/0/surfaceFieldValue.dat")) and
os.path.exists(os.path.join(dirs, "postProcessing/avgsurf_out/0/surfaceFieldValue.dat")):
        global p_in
        p_in = StringVar()
        p_in.set("")
        global p_out
        p_out = StringVar()
        p_out.set("")
        path201 = os.path.join(dirs, "postProcessing/avgsurf_in/0/surfaceFieldValue.dat")
        path301 = os.path.join(dirs, "postProcessing/avgsurf_out/0/surfaceFieldValue.dat")
        with open(path201) as f:
            reader = csv.reader(f, delimiter="\t")
            for line in reader:
                print(line)
        p_in = float(line[1])
        with open(path301) as f:
            reader = csv.reader(f, delimiter="\t")
            for line in reader:
                print(line)

        p_out = float(line[1])
        p_drop_e.delete(0, END)
        a_res_e.delete(0, END)
        p_drop_e.insert("insert",(p_in - p_out)/1000)
        a_res_e.insert("insert",(p_in - p_out) / (1000 * float(pvfr.get())))

    ##
    if os.path.exists(os.path.join(dirs, "p_cut_1.png")):
        if os.path.exists(os.path.join(dirs, "v_cut_2.png")):
            status_e.delete(0, END)
            status_e.insert(0, "completed")
    ##
    save_button = Button(root, text = "Save", font=('Times_New_Roman', 16), command=save_win5,
activebackground='blue').place(x = 300, y = 510)
    back_button = Button(root, text = "Back", font=('Times_New_Roman', 16), command=back_win5,
activebackground='blue')
    back_button.pack(side=BOTTOM)

##########################################################################################
# Next button on tab 1
def next_win1():
    for widgets in root.winfo_children():
        widgets.destroy()

    tab2()

##########################################################################################
# Back button on tab 2
def back_win2():
    for widgets in root.winfo_children():
        widgets.destroy()

    tab1()

# Next button on tab 2
def next_win2():
    if un.get() and pn.get() and pa.get() and pd.get() and fn.get():
        username = un.get()
        patientname = pn.get()
        patientage = pa.get()
        patientdoctor = pd.get()
        filename = fn.get()
        file1 = open("fn.txt","w")
        file1.write(filename)
        file1.close()
        if username and patientname and patientage and patientdoctor and filename:
            global path
            global dirs
            path = StringVar()
            dirs = StringVar()
            dirs.set("")
            path_ = askdirectory()
            path.set(path_)
            dirs = os.path.join(path.get(), fn.get())
            file1 = open("sdir.txt","w")
            file1.write(dirs)
            file1.close()
            if not os.path.exists(dirs):
                os.makedirs(dirs)
                messagebox.showinfo('Info','Folder name created successfully!')
            else:
                messagebox.showinfo("Info",'for you Info the folder name already exists')
            filepath = "gui_data.xlsx"
            if not os.path.exists(filepath):
                wb1 = openpyxl.Workbook()
                sh1 = wb1.create_sheet('tab1')
                heading = ["Date","User Name","Patient Name","Patient Age","Patient Doctor", "File
Name", "File Location", "Status pre-proc","Geo chk","Patient VFR","Status CFD","Status Post-
proc","Pressure drop (kPa)","Airway resistance (cmH20/L/S)"]
                sh1.append(heading)
                wb1.save(filepath)
                wb1 = openpyxl.load_workbook(filepath)
                ws = wb1['tab1']
                ws.append([os.popen('date').read(), username, patientname, patientage, patientdoctor,
filename])
                wb1.save(filepath)
            else:
                wb1 = openpyxl.load_workbook(filepath)
                ws = wb1['tab1']
                ws.append([os.popen('date').read(), username, patientname, patientage, patientdoctor,
filename])
                wb1.save(filepath)
        for widgets in root.winfo_children():
            widgets.destroy()

        tab3()

    else:
        messagebox.showerror("Error", "Missing information")
```

```
##########################################################################################
# Back button on tab 3
def back_win3():
    for widgets in root.winfo_children():
        widgets.destroy()

    tab2()

# Next button on tab 3
def next_win3():
    if cvar.get() == 1:
        if spp.get() == "completed":
            if os.path.exists(os.path.join(dirs, "geo_in.txt")) and os.path.exists(os.path.join(dirs,
"blendout.jpg")):
                filepath = "gui_data.xlsx"
                wb1 = openpyxl.load_workbook(filepath)
                ws = wb1['tab1']
                ws.append([os.popen('date').read(), un.get(), pn.get(), pa.get(), pd.get(), fn.get(),
file_path_var.get(), spp.get()])
                wb1.save(filepath)
                for widgets in root.winfo_children():
                    widgets.destroy()

                tab4()
            else:
                messagebox.showerror("Error", "Missing Geo Location")
        else:
            messagebox.showerror("Error", "Geo status should be completed")
    else:
        messagebox.showerror("Error", "Please check the image")

##########################################################################################
# Back button on tab 4

def back_win4():
    for widgets in root.winfo_children():
        widgets.destroy()

    tab3()

# Next button on tab 4
def next_win4():
    if scfd.get() == "completed":
        filepath = "gui_data.xlsx"
        wb1 = openpyxl.load_workbook(filepath)
        ws = wb1['tab1']
        ws.append([os.popen('date').read(), un.get(), pn.get(), pa.get(), pd.get(), fn.get(),
file_path_var.get(), spp.get(), pvfr.get(), scfd.get()])
        wb1.save(filepath)
        for widgets in root.winfo_children():
            widgets.destroy()

        tab5()
    else:
        messagebox.showerror("Error", "CFD status should be completed")

#########################################################################################
# Back button on tab 5
def back_win5():
    for widgets in root.winfo_children():
        widgets.destroy()

    tab4()

# Next button on tab 5
def save_win5():
    if spop.get() == "completed":
        if pdr.get() and arr.get():
            filepath = "gui_data.xlsx"
            wb1 = openpyxl.load_workbook(filepath)
            ws = wb1['tab1']
            ws.append([os.popen('date').read(), un.get(), pn.get(), pa.get(), pd.get(), fn.get(),
file_path_var.get(), spp.get(), pvfr.get(), scfd.get(), spop.get(), pdr.get(), arr.get()])
            wb1.save(filepath)
        else:
            messagebox.showerror("Error", "Values are Empty")
    else:
        messagebox.showerror("Error", "Please complete Post-processing to save")

tab1()

root.mainloop()
```

Paraview script:

```
# Msc Ortho CFD paraview 5.12 script File
# Computational Fluid Dynamics Lab
# Author = Uday Tummala
# Dr. Carlos F. Lange
# Department of Mechanical Engineering
# University of Alberta
#
# Date 2024-01-07
# cmd opens gui = paraview --script=home/uday/Desktop/msc_ortho/paraview_ortho.py
# cmd for local = pvbatch paraview_ortho.py
# cmd for server no GUI = pvpython paraview_ortho.py

import paraview
import time
import os

# version details
paraview.compatibility.major = 5
paraview.compatibility.minor = 12

from paraview.simple import *

# disable automatic camera reset on 'Show'
paraview.simple._DisableFirstRenderCameraReset()
start_time = time.time()

###########################################################################################
# Open case.foam file
file = open("sdir.txt", "r")
path1 = file.readline()
file.close()
path111 = os.path.join(path1, "case.foam")
casefoam = OpenFOAMReader(registrationName='case.foam', FileName=path111)
casefoam.MeshRegions = ['internalMesh']
casefoam.CellArrays = ['U', 'p']

###########################################################################################
# Define parameters
animationScene1 = GetAnimationScene()
animationScene1.UpdateAnimationUsingDataTimeSteps()
renderView1 = GetActiveViewOrCreate('RenderView')
casefoamDisplay = Show(casefoam, renderView1, 'UnstructuredGridRepresentation')
materialLibrary1 = GetMaterialLibrary()
casefoamDisplay.SetScalarBarVisibility(renderView1, True)
casefoam = GetActiveSource()
casefoamDisplay = GetDisplayProperties(casefoam, view=renderView1)
casefoamDisplay.RescaleTransferFunctionToDataRange(True, False)
```

```
layout1 = GetLayout()
layout1.SetSize(1577, 733)

#########################################################################################
# pressure & velocity pic-1 export
ColorBy(casefoamDisplay, ('POINTS', 'p'))
renderView1.InteractionMode = '2D'
pLUT = GetColorTransferFunction('p')
pPWF = GetOpacityTransferFunction('p')
pLUT.ApplyPreset('Rainbow Uniform', True)
pTF2D = GetTransferFunction2D('p')
renderView1.Update()
casefoamDisplay.SetScalarBarVisibility(renderView1, True)
pLUTColorBar = GetScalarBar(pLUT, renderView1)
pLUTColorBar.Orientation = 'Horizontal'
pLUTColorBar.WindowLocation = 'Any Location'
pLUTColorBar.Position = [0.145, 0.019]
pLUTColorBar.ScalarBarLength = 0.34

# current camera placement for renderView1
renderView1.CameraPosition = [0.193, -0.0203, -0.0642]
renderView1.CameraFocalPoint = [0.0654, 0.0892, 0.0417]
renderView1.CameraViewUp = [0.514, -0.205, 0.833]
renderView1.CameraParallelScale = 0.0515
path2 = os.path.join(path1, "p_full_1.png")
SaveScreenshot(path2, renderView1, 16, ImageResolution=[1577, 733],
OverrideColorPalette='WhiteBackground')
ColorBy(casefoamDisplay, ('POINTS', 'U', 'Magnitude'))
HideScalarBarIfNotNeeded(pLUT, renderView1)
casefoamDisplay.SetScalarBarVisibility(renderView1, True)
uLUT = GetColorTransferFunction('U')
uPWF = GetOpacityTransferFunction('U')
uTF2D = GetTransferFunction2D('U')
uLUT.ApplyPreset('Rainbow Uniform', True)
renderView1.Update()
casefoamDisplay.SetScalarBarVisibility(renderView1, True)
uLUTColorBar = GetScalarBar(uLUT, renderView1)
uLUTColorBar.Orientation = 'Horizontal'
uLUTColorBar.WindowLocation = 'Any Location'
uLUTColorBar.Position = [0.145, 0.019]
uLUTColorBar.ScalarBarLength = 0.34
casefoamDisplay.RescaleTransferFunctionToDataRange(True, False)
path3 = os.path.join(path1, "v_full_1.png")
SaveScreenshot(path3, renderView1, 16, ImageResolution=[1577, 733],
OverrideColorPalette='WhiteBackground')
```

```
#######################################################################################
# pressure pic-2,3,4,5 export
ColorBy(casefoamDisplay, ('POINTS', 'p'))
HideScalarBarIfNotNeeded(uLUT, renderView1)
casefoamDisplay.SetScalarBarVisibility(renderView1, True)
renderView1.ResetActiveCameraToPositiveX()
renderView1.ResetCamera(False, 0.9)
renderView1.CameraPosition = [-0.1596, 0.0083, 0.0482]
renderView1.CameraFocalPoint = [0.0392, 0.0083, 0.049]
renderView1.CameraViewUp = [0.0, 0.0, 1.0]
renderView1.CameraParallelScale = 0.05146098720456678
casefoamDisplay.RescaleTransferFunctionToDataRange(True, False)
path4 = os.path.join(path1, "p_full_2.png")
SaveScreenshot(path4, renderView1, 16, ImageResolution=[1577, 733],
OverrideColorPalette='WhiteBackground')
renderView1.ResetActiveCameraToNegativeX()
renderView1.ResetCamera(False, 0.9)
renderView1.CameraPosition = [0.33, 0.096, 0.05]
renderView1.CameraFocalPoint = [0.039, 0.096, 0.05]
renderView1.CameraViewUp = [0.0, 0.0, 1.0]
renderView1.CameraParallelScale = 0.052
casefoamDisplay.RescaleTransferFunctionToDataRange(True, False)
path5 = os.path.join(path1, "p_full_3.png")
SaveScreenshot(path5, renderView1, 16, ImageResolution=[1577, 733],
OverrideColorPalette='WhiteBackground')
renderView1.ResetActiveCameraToPositiveY()
renderView1.ResetCamera(False, 0.9)
renderView1.CameraPosition = [0.072, -0.244, 0.0494]
renderView1.CameraFocalPoint = [0.073, 0.0469, 0.0498]
renderView1.CameraViewUp = [0.0, 0.0, 1.0]
renderView1.CameraParallelScale = 0.0518
casefoamDisplay.RescaleTransferFunctionToDataRange(True, False)
path6 = os.path.join(path1, "p_full_5.png")
SaveScreenshot(path6, renderView1, 16, ImageResolution=[1577, 733],
OverrideColorPalette='WhiteBackground')
renderView1.ApplyIsometricView()
renderView1.ResetCamera(False, 0.9)
renderView1.CameraPosition = [0.177, 0.25, 0.22]
renderView1.CameraFocalPoint = [0.01, 0.082, 0.0497]
renderView1.CameraViewUp = [-0.3, -0.51, 0.81]
renderView1.CameraParallelScale = 0.063
casefoamDisplay.RescaleTransferFunctionToDataRange(True, False)
path7 = os.path.join(path1, "p_full_4.png")
SaveScreenshot(path7, renderView1, 16, ImageResolution=[1577, 733],
OverrideColorPalette='WhiteBackground')
```

```
##########################################################################################
# pressure & velocity section view pic-1 export
clip1 = Clip(registrationName='Clip1', Input=casefoam)
clip1.ClipType = 'Plane'
clip1.HyperTreeGridClipper = 'Plane'
clip1.Scalars = ['POINTS', 'p']
clip1.Value = 24.54
clip1.HyperTreeGridClipper.Origin = [0.039, 0.0468, 0.05385]
clip1.ClipType.Origin = [0.0364, 0.0468, 0.0538]
renderView1.Update()
clip1Display = Show(clip1, renderView1, 'UnstructuredGridRepresentation')
Hide(casefoam, renderView1)
ColorBy(clip1Display, ('POINTS', 'p'))
clip1Display.RescaleTransferFunctionToDataRange(True, False)
clip1Display.SetScalarBarVisibility(renderView1, True)
HideInteractiveWidgets(proxy=clip1.ClipType)
renderView1.CameraPosition = [0.2033, 0.0156, 0.0884]
renderView1.CameraFocalPoint = [0.0598, 0.0873, 0.0523]
renderView1.CameraViewUp = [-0.18498, 0.12046, 0.975]
renderView1.CameraParallelScale = 0.052
path8 = os.path.join(path1, "p_cut_1.png")
SaveScreenshot(path8, renderView1, 16, ImageResolution=[1577, 733],
OverrideColorPalette='WhiteBackground')
ColorBy(clip1Display, ('POINTS', 'U', 'Magnitude'))
HideScalarBarIfNotNeeded(pLUT, renderView1)
clip1Display.RescaleTransferFunctionToDataRange(True, False)
clip1Display.SetScalarBarVisibility(renderView1, True)
path9 = os.path.join(path1, "v_cut_1.png")
SaveScreenshot(path9, renderView1, 16, ImageResolution=[1577, 733],
OverrideColorPalette='WhiteBackground')

##########################################################################################
# pressure & velocity section view pic-2 export
clip1.ClipType.Origin = [0.04306, 0.0468, 0.05385]
ColorBy(clip1Display, ('POINTS', 'U', 'Magnitude'))
renderView1.Update()
renderView1.CameraPosition = [0.2381, 0.10142, 0.0492]
renderView1.CameraFocalPoint = [0.0216, 0.10142, 0.0493]
renderView1.CameraViewUp = [0.0, 0.0, 1.0]
renderView1.CameraParallelScale = 0.056
path10 = os.path.join(path1, "v_cut_2.png")
SaveScreenshot(path10, renderView1, 16, ImageResolution=[1577, 733],
OverrideColorPalette='WhiteBackground')
ColorBy(clip1Display, ('POINTS', 'p'))
HideScalarBarIfNotNeeded(uLUT, renderView1)
clip1Display.RescaleTransferFunctionToDataRange(True, False)
clip1Display.SetScalarBarVisibility(renderView1, True)
pLUT.RescaleTransferFunction(-13, 66)
pPWF.RescaleTransferFunction(-13, 66)
```

```
path11 = os.path.join(path1, "p_cut_2.png")
SaveScreenshot(path11, renderView1, 16, ImageResolution=[1577, 733],
OverrideColorPalette='WhiteBackground')

####################################################################################
# rescale color and/or opacity maps used to exactly fit the current data range
clip1Display.RescaleTransferFunctionToDataRange(False, True)
Delete(clip1)

####################################################################################
# export velocity plot with streamlines pic 1
SetActiveSource(casefoam)
casefoamDisplay = Show(casefoam, renderView1, 'UnstructuredGridRepresentation')
ColorBy(casefoamDisplay, ('FIELD', 'vtkBlockColors'))
casefoamDisplay.Opacity = 0.30
streamTracer1 = StreamTracer(registrationName='StreamTracer1', Input=casefoam,
   SeedType='Line')
streamTracer1.Vectors = ['POINTS', 'U']
streamTracer1.MaximumStreamlineLength = 0.12;
streamTracer1.SeedType.Point1 = [5.0e-05, -0.0003, 0.01]
streamTracer1.SeedType.Point2 = [0.08, 0.1, 0.1]
streamTracer1Display = Show(streamTracer1, renderView1, 'GeometryRepresentation')
streamTracer1Display.SetScalarBarVisibility(renderView1, True)
ColorBy(streamTracer1Display, ('POINTS', 'U', 'Magnitude'))
HideScalarBarIfNotNeeded(pLUT, renderView1)
streamTracer1Display.RescaleTransferFunctionToDataRange(True, False)
streamTracer1Display.SetScalarBarVisibility(renderView1, True)
renderView1.Update()
tube1 = Tube(registrationName='Tube1', Input=streamTracer1)
tube1.Scalars = ['POINTS', 'p']
tube1.Vectors = ['POINTS', 'Normals']
tube1.Radius = 0.0004
tube1Display = Show(tube1, renderView1, 'GeometryRepresentation')
Hide(streamTracer1, renderView1)
tube1Display.SetScalarBarVisibility(renderView1, True)
ColorBy(tube1Display, ('POINTS', 'U', 'Magnitude'))
HideScalarBarIfNotNeeded(pLUT, renderView1)
tube1Display.RescaleTransferFunctionToDataRange(True, False)
tube1Display.SetScalarBarVisibility(renderView1, True)
tube1Display.RescaleTransferFunctionToDataRange(False, True)
renderView1.Update()
uLUTColorBar = GetScalarBar(uLUT, renderView1)
uLUTColorBar.Orientation = 'Horizontal'
uLUTColorBar.WindowLocation = 'Any Location'
uLUTColorBar.Position = [0.152, 0.022]
uLUTColorBar.ScalarBarLength = 0.34
uLUT.RescaleTransferFunction(0.0, 12.026)
uPWF.RescaleTransferFunction(0.0, 12.026)
```

```
########################################################################################
# export velocity plot with streamlines pic 2
SetActiveSource(casefoam)
streamTracer2 = StreamTracer(registrationName='StreamTracer2', Input=casefoam,
    SeedType='Line')
streamTracer2.Vectors = ['POINTS', 'U']
streamTracer2.MaximumStreamlineLength = 0.0942
streamTracer2.SeedType.Point1 = [5.0e-05, -0.0003, 0.01]
streamTracer2.SeedType.Point2 = [0.08, 0.1, 0.1]
streamTracer2.MaximumStreamlineLength = 0.12
streamTracer2Display = Show(streamTracer2, renderView1, 'GeometryRepresentation')
streamTracer2Display.SetScalarBarVisibility(renderView1, True)
ColorBy(streamTracer2Display, ('POINTS', 'U', 'Magnitude'))
streamTracer2Display.RescaleTransferFunctionToDataRange(True, False)
streamTracer2Display.SetScalarBarVisibility(renderView1, True)
renderView1.Update()
tube2 = Tube(registrationName='Tube2', Input=streamTracer2)
tube2.Scalars = ['POINTS', 'p']
tube2.Vectors = ['POINTS', 'Normals']
tube2.Radius = 0.0004
tube2Display = Show(tube2, renderView1, 'GeometryRepresentation')
Hide(streamTracer2, renderView1)
ColorBy(tube2Display, ('POINTS', 'U', 'Magnitude'))
HideScalarBarIfNotNeeded(pLUT, renderView1)
tube2Display.RescaleTransferFunctionToDataRange(True, False)
tube2Display.SetScalarBarVisibility(renderView1, True)
renderView1.Update()
uLUTColorBar.Position = [0.139, 0.034]
uLUTColorBar.ScalarBarLength = 0.34
uLUT.RescaleTransferFunction(0.0, 12.026)
uPWF.RescaleTransferFunction(0.0, 12.026)
casefoamDisplay = Show(casefoam, renderView1, 'UnstructuredGridRepresentation')
renderView1.InteractionMode = '2D'
renderView1.ResetActiveCameraToPositiveY()
renderView1.ResetCamera(False, 0.9)
renderView1.CameraPosition = [0.072, -0.244, 0.0494]
renderView1.CameraFocalPoint = [0.073, 0.0469, 0.0498]
renderView1.CameraViewUp = [0.0, 0.0, 1.0]
renderView1.CameraParallelScale = 0.0518
path12 = os.path.join(path1, "v_streamlines1.png")
SaveScreenshot(path12, renderView1, 16, ImageResolution=[1577, 733],
OverrideColorPalette='WhiteBackground')

# current camera placement for renderView1
renderView1.InteractionMode = '3D'
renderView1.CameraPosition = [0.216, -0.027, -0.02]
renderView1.CameraFocalPoint = [0.0663, 0.0866, 0.0473]
renderView1.CameraViewUp = [0.272, -0.2, 0.942]
renderView1.CameraParallelScale = 0.0513
```

```
path13 = os.path.join(path1, "v_streamlines2.png")
SaveScreenshot(path13, renderView1, 16, ImageResolution=[1577, 733],
OverrideColorPalette='WhiteBackground')

################################################################################
# Quit Paraview
print("Finished taking photos of CFD Model in Paraview_v5.12")
print("Please check output images p & v .png files")
print("Time taken for postprocessing: ",time.time()-start_time)
```