Automated workflow for CFD analysis in the aorta: Additional information dossier

L.M. Thomas Y. Brunt, J. Bakkum, P. Jongepier

(5305748), (5322669), (5086361), (4926684)

Technische Universiteit Delft

Delft, June 14, 2024

Supervisor: S. Pirola

Contents

1

Python & Git

All the code written for this research can be accessed through the GitHub repository at https://github.com/lmthomasTU/Aortic_CFD_workflow.

1.1. Usecase example & manual

The usage of the program will be explained for an example input. Data used for this example are case nr. 163 of the Romero et al. [1] dataset (available at https://www.uv.es/commlab/blog-details-3DAORTAMODEL.html or in the Aortic CFD GitHub) and velocity profile 001 from [2] (available at https://zenodo.org/records/7833575 or in the Aortic CFD GitHub.

1.1.1. software and packages

Main software:

- Download the contents of the main branch from the GitHub repository linked above and unzip to any location.

External programs:

- FEBio 4.5.0 https://febio.org/downloads/
- FEBio Studio 2.5.0
- Python 3.12.3 https://www.python.org/downloads/

Python packages (may be installed via `pip install` or with Anaconda):

- pymmg
- numpy
- pyvista
- tetgen

1.1.2. Input files

For the velocity profiles it does not matter how they are placed, as long as all 20 timepoints are within the same folder. Each geometry must be placed as follows:

```
<Folder_to_select>/<case_folder>/meshes/inlet.stl outlet.stl wall.stl
```

where `<Folder_to_select>` contains a sub-folder for each geometry to be run, in this case only one. Names in <> can be arbitrary.

1.1.3. Program settings

At the top of the file `main_workflow` all of the necessary settings for FEBio and preprocessing can be found. Explanations for all settings can be seen in the code comments.

The settings used in this example are as follows:

```
FEBio_parameters = dict(
    hr = 100,                                       #heartrate in bpm for cycle time

    #outputs of the FEBio simulation, set all required data to True
    displacement = False,
    fluid_pressure = False,
    nodal_fluid_velocity = False,
    fluid_stress = True,
    fluid_velocity = True,
    fluid_acceleration = False,
    fluid_vorticity = False,
    fluid_rate_of_deformation = False,
    fluid_dilatation = False,
    fluid_volume_ratio = False,

    vtk = True,                                     #Output type, set to True to use postprocessing script,
                                                    #set to false to view the output in FEBio studio
    #fluidconstants
    materialtype = 'fluid',
    density = 1000,                                 #kg/m^3
    k = 2200000,                                    #bulkmodulus
    viscoustype = "Newtonian fluid",
    kappa = 1,                                      #bulk viscosity
    mu = 0.056,                                     #shear viscosity)

    #simulationcontrol
    time_steps = 600,                              #initial time steps, changes towards dtmax
    step_size = 0.001,                              #seconds
    dtmin = 0,                                      #min timestepsize
    dtmax = 0.01,                                   #max timestepsize

    #loadcontroller interpolation
    interpolate = 'LINEAR')                         #can be'STEP',  'LINEAR' or 'SMOOTH'

#Path for FEBio solver executable
FEBio_path = r"C:/Program Files/bin/febio4.exe"                 #Path 1
#FEBio_path = r"C:/Program Files/FEBioStudio2/bin/febio4.exe"   #Path 2

#Meshing parameters
max_retry = 2               #Maximum number of retries for wall surface remeshing if quality is too low after volume mesh

mmg_parameters = {          #Surface meshing parameters
    'mesh_density': '0.1',  #hausdorf parameter of mmg, defines amount of added detail at curvature
    'sizing': '1',          #forces similarly sized poly's, legacy option (used for stitching), can possibly be dropped
    'detection angle': '45'}#Sharp angle detection of mmg, preserves the edges at the in- and outlet

mmg3d_parameters = {        #Volume meshing parameters
    'hausd': '0.1',
    'detection angle': '45'}

mmg3d_sol_parameters = {    #Paremeters for local mesh refinement (used to create an extra fine boundary layer)
    'bl_thickness': 1,
    'bl_edgelength': 1,     #Max edgelength within the boundary layer
    'edgelength': 15}       #Max edgelength in the interior

tetgen_parameters = dict(   #Parameters for initial volume mesh, not very important, as long as the mesh is finer than
     the bl_thickness
    order=1,
    mindihedral=20,
    minratio=1.5,
    nobisect=True,
    fixedvolume=True,
    maxvolume=1)            #Controlls the density

#Run qualifications (case will be discarded if not met)
max_elements = 1000000
min_jacobian = 0.1
max_aspect   = 5

#Angle for identification of surfaces
id_angle = 35

#Mapping interpolation options
intp_options = {
    'zero_boundary_dist': 0.2,  # percentage of border with zero velocity (smooth damping at the border)
    'zero_backflow': False,     # set all backflow components to zero
    'kernel': 'linear',         # RBF interpolation kernel (linear is recommended)
    'smoothing': 0.5,           # interpolation smoothing, range recommended [0, 2]
    'degree': 0,                # degree of polynomial added to the RBF interpolation matrix
    'hard_noslip': False}       # check if no-slip condition on walls is met

#Plotting boolean, when True: code generates intermediate plots of workflow
show_plot = True
```

1.1.4. Running the workflow

Open the file `main_workflow` in your Python editor of choice and change the settings to match the ones listed above. Make sure the setting `FEBio_path` matches the location of the FEBio solver (febio4 when writing this example) on your machine. Next, run the file.

Three popups should now appear one by one, the first one asks for the location of the input geometries (shown in Figure 1.1, the second asks for the velocity profiles, and the last one asks for an output location. The output location can be anywhere on your computer.

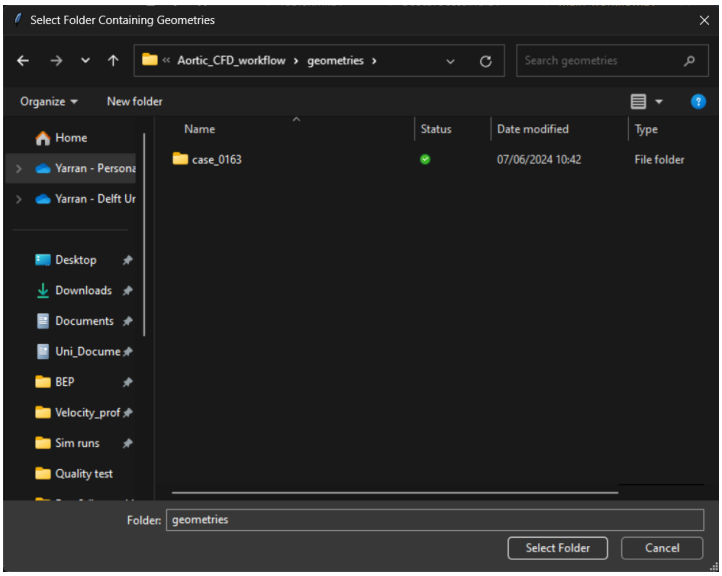


Figure 1.1: *Folder selection popup with the correct selection*

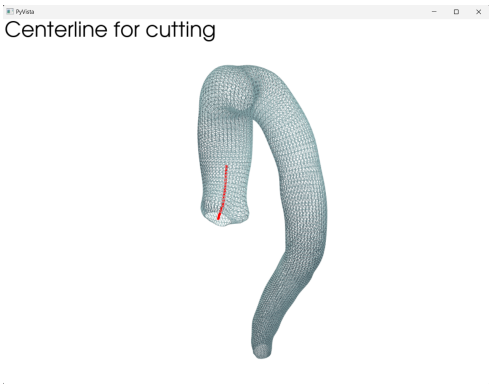
Once all inputs and the output location are selected, the program will start processing the geometries. As plotting has been set to `True` in the settings, plots will be shown of each processing step. To make the program run faster and without user intervention, disable this option, however to check if everything is set up correctly or for debugging it can be quite useful.

At this point the code will automatically perform the following actions:

1. Cutting away the aortic root
2. Generate caps to close the geometry
3. Remesh the surface
4. Convert triangular surf. mesh to tetrahedal volume mesh
5. Identify regions close to the aortic wall
6. Remesh to improve quality and to provide a higher density near the wall
7. Identify boundary condition surfaces
8. Map the input profiles to the inlet surface
9. Store all generated data in an FEBio job file
10. Run FEBio and store the results as .VTK files for each timestep

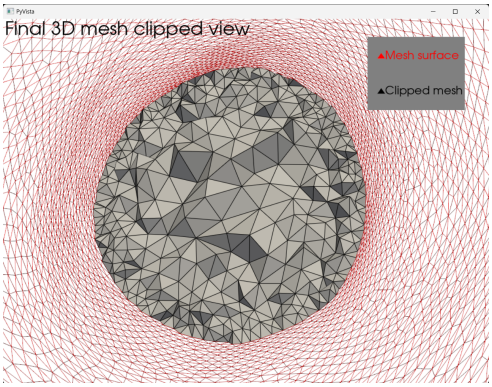
To check your output, a few of the plots are included in Figure 1.2. Once all plots have been shown FEBio will start. At each step the code will report its progress in the command line, if pre-processing has been completed successfully, the message shown in Figure 1.3 should be printed. All processing between folder selection and the start of the solver takes about ninety seconds, plotting excluded, on the hardware used [1].

[1] Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz, 16GB RAM

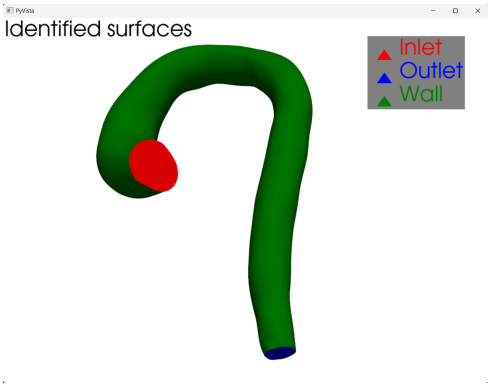


(a) *Centerline detection*

(b) *Completed surface after cutting & capping*



(c) *Clipped view of the final volume mesh*



(d) *Identified surfaces for boundary conditions*

Figure 1.2: *Identified cutting location and resulting geometry*

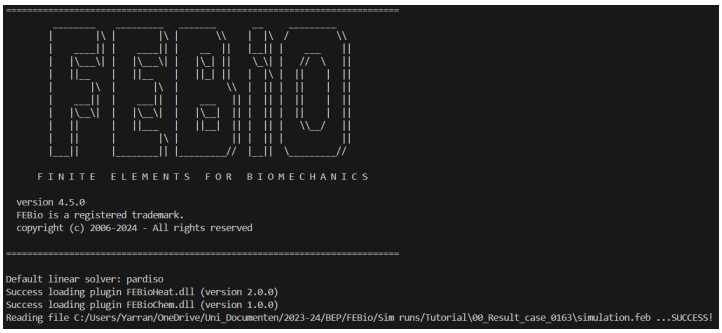


Figure 1.3: *Start of the FEBio solver*

Once the solver has finished (takes approximately 40 minutes), the output directory will be populated with the mapped velocity profiles, a set of .vtk files, the .feb file used by FEBio to start the simulation, and a simulation.txt file written by FEBio. The text file contains a report of the simulation settings and convergence, at the bottom you can find information about the running time.

To view the vtk files, select the `postprocessing` file in the python editor. Make sure the command at the bottom of the file is `wss_cut_and_hist(100)` and run. A popup will appear asking to select files for postprocessing. Any amount .vtk files can be selected, also from different simulation runs. For this example we will only select `simulation21.vtk` as this snapshot is around the systolic peak. The postprocessing script will cut the geometry at the horizontal point to isolate the ascending aorta, plot a visualisation of the wall shear stress (Figure 1.4), and print quantitative information in the command line (Figure 1.5).

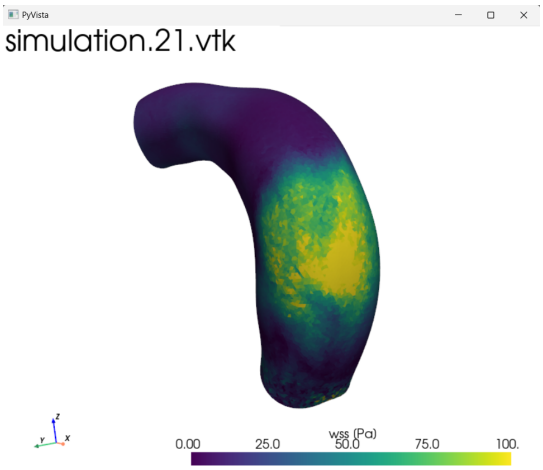


Figure 1.4: *WSS plot of timestep 21*

```
Histogram of wss for: simulation.21.vtk
(Percentile: Value[Pa])
99: 104.32755859374981
98: 92.51075744628909
96: 81.40122833251955
94: 74.10884155273435
92: 67.24194976806649
90: 59.06098480224613
```

Figure 1.5: *Numerical output*

2

Processing tools

2.1. Centerline

First step after importing of the geometry is performing a cut to exclude the aortic root. To achieve this, the centerline needs to be extracted. Doing this computationally is a complex process. Many papers have been published on the concept of medial skeleton extraction [3][4], but not many of these methods have been elaborated into an open access software or are capable to handle 3D-structures. External programs, such as the VMTK toolbox [5], make it possible to perform centerline extraction, but this requires installing dependencies outside of the Python environment. For this reason and the fact that a high cutting accuracy is not needed, an algorithm based on iterative mesh slicing (IMS) has been written specifically for this thesis.

IMS makes use of two attributes of the aorta dataset to reduce the complexity of the problem:

1. The center point of the inlet contour and direction of the centerline are known
2. The vessel only has one in- and outlet (a crucial requirement for the algorithm to work)

From this starting point the method walks along the center of the vessel and relies on finding the geometric center of slices of the wall to adjust its course. The exact steps can be seen in Figure 2.1. In the first step, the normal of the inlet surface is followed along a distance of 1mm (length of the normal vector) to point (1). The distance is based on the location of the data of the inflow profiles, as explained later with the boundary conditions. At step 2 the geometry is cut perpendicular to the vector from point (0) to point (1) to obtain a planar surface. In step three, the geometric center of the cutting plane (2) is calculated, and point (3) is defined following the vector from point (0) to point (2). This process is repeated until a set length of center line is found. The output of the algorithm consists of points (0, 2, 4) etc.

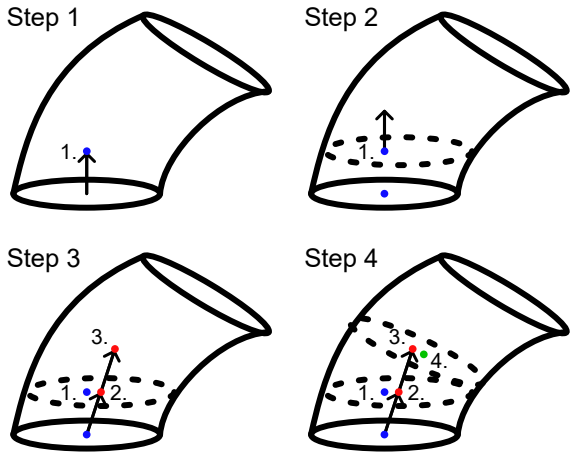


Figure 2.1: *Visual representation of the IMS algorithm*

To test the validity of the generated centerline a comparison to an established centerline generation tool is used. VMTK is chosen for this purpose as it is widely used in the area of vascular modelling [6][7] [8]. A test geometry (shown in Figure 2.2) with a known centerline was created loosely based on the geometries used by Zoubir et al. [9]. Both VMTK and IMS centerlines where generated through the entire length of the mesh. The

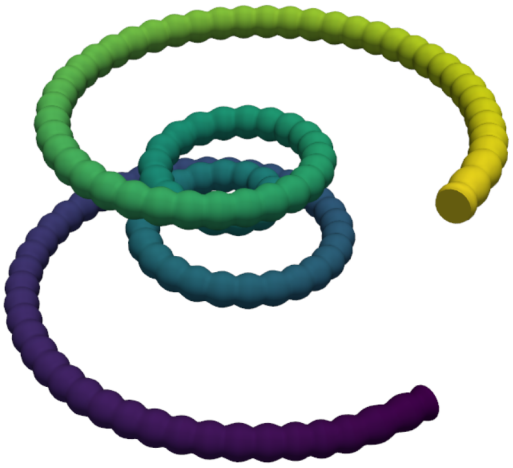


Figure 2.2: *Test geometry*

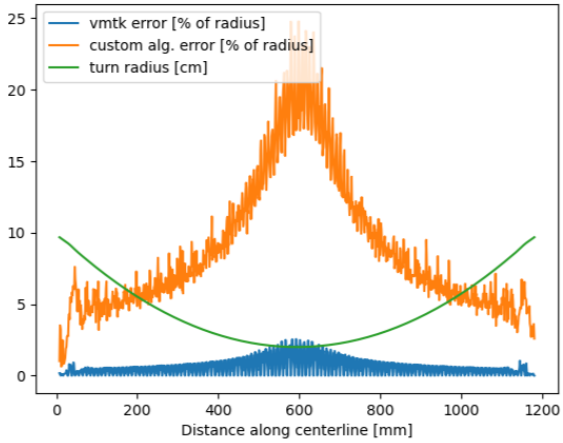


Figure 2.3: *Error as % of local radius for VMTK and IMS*

resulting error can be seen in Figure 2.3. On the more curved sections where VMTK maintains a relative error of less than a few percentage points, the IMS algorithm quickly loses accuracy.

When evaluated on one of the aorta geometries IMS performs two times better than on the test geometry. Benchmarked against a VMTK centerline, the error remains lower than ten percent of the radius for most of the datapoints, as can be seen in Figure 2.4. Most importantly for application in this research, IMS performs sufficiently in the first few centimeters: the area where cutting will take place.

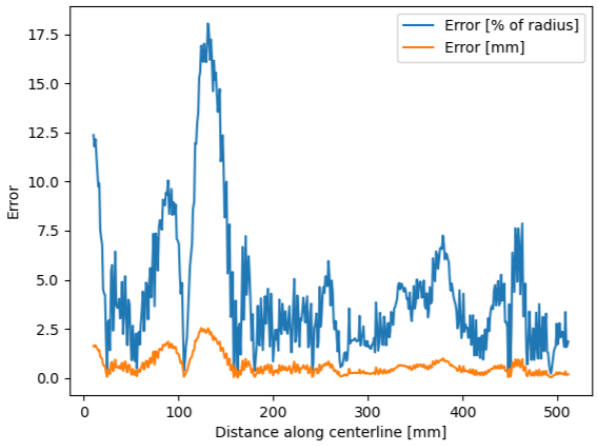


Figure 2.4: *Difference between IMS and VMTK centerlines as % of approximate local radius*

2.2. Cutting

The geometry provided by Romero et al. [1] starts at the base of the aortic root, whereas the velocity profiles from Saitta at al. are collected at a location chosen in refrence to the pulmonary artery bifurcation point just above the aortic root [2]. As the pulmonary artery is no longer present in the Romero et al. dataset, the end of the aortic root is chosen as the cutting point to match the point of data acquisition as closely as possible while also cutting in a consistent place.

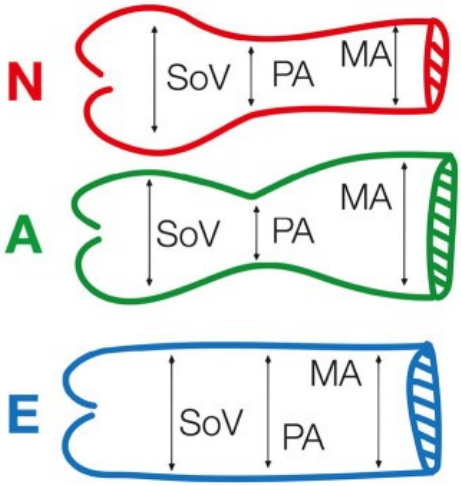
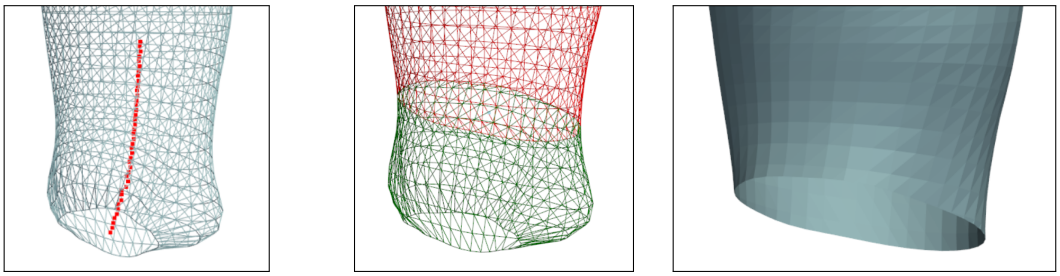


Figure 2.5: *Aorta phenotypes. Adapated from [10]*

Schaefer et al. [10] identifies three different aorta phenotypes, based on the shape aortic root and ascending aorta, shown in Figure 2.5 To perform the cut in a consistent location at the end of the aortic root, for phenotypes N and A, the slice n is selected where the cutting surface area A_n has a local minimum or, if that point does not exist (most commonly in aortas of phenotype E), the cut is made where the rolling average of the derivative of the surface area to the slice number is minimal 2.1.

$$n = min(A_n) \bigvee min\left(\frac{\delta(A_{n-1} + A_n + A_{n+1})}{3\delta n}\right) \tag{2.1}$$

Using this method, the end of the aortic root can be consistently identified. In Figure 2.6 the results of the algorithm at work can be seen.



(a) *Centerline* **(b)** *Identified cutting location* **(c)** *Cut geometry*

Figure 2.6: *Identified cutting location and resulting geometry*

Next, to be able to provide a closed geometry for the 3D meshing software, the vessel must be capped. This operation is performed in three steps. First the edges of the geometry are detected. Next closed loops are extracted one by one. These loops, consisting of line segments, are then converted to surfaces by adding a single point in the middle of the loop and adding that point to the connectivity of each line segment, creating triangles. One of the extracted loops and the generated cap are visualized in Figure 2.7.

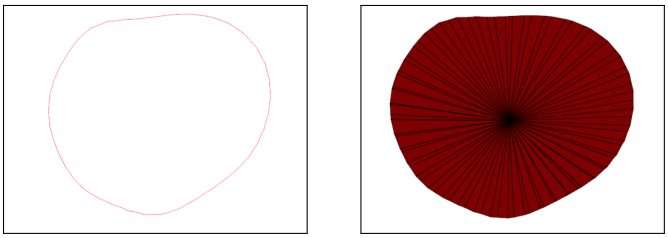


Figure 2.7: *Loop (L) and generated cap (R)*

2.2.1. Cutting validation

The data collected for the cutting validation can be found in table 2.1

Table 2.1: *Cutting location difference between IMS and VMTK*

Error [mm]	**Mean**
0,711	1,1873
1,092	**Std. Deviation**
1,645	0,565
0,663	
1,144	
1,001	
1,568	
2,446	
1,014	
0,589	

2.3. Surface meshing

Many metrics exist to asses the quality of a mesh. Examples are mesh smoothness, cell aspect ratio and mean scaled jacobian. However, the numerical output of these metrics tells us little about the actual suitability of the mesh for a computational process, as this is highly dependent on the specific application [11]. To properly determine if the quality of a generated mesh is sufficient to run a meaningful simulation, these metrics need to be applied as an aid alongside experience with the CFD process itself.

A situation where quality metrics can be used more directly is when comparing different algorithms or algorithm specific parameters. To choose the most suitable candidate for the surface meshing software, a

python script was written to generate a quantitative analysis of mesh quality.

The script assesses meshes in two different ways:

- Similarity of the remeshed surface mesh in comparison to the original mesh.
- The quality of the elements of the remeshed surface

For the first assessment, a comparison must be made between the original input mesh and the remeshed surface. To accomplish this, a script is written that calculates for each input node the smallest distance to the output mesh surface. The mean of all the values is the *approximate similarity score S*, which is used to evaluate the similarity between the meshes.

$$S = \frac{1}{n}\sum_{i=1}^{n} d_i \tag{2.2}$$

$$S = \textit{approximate similartiy}, \quad d = \textit{node-to-surface distance}$$

Engvall et al. proposes the mean scaled Jacobian as a comprehensive measure for mesh quality of triangular surface meshes. The mean scaled Jacobian is a measure of the ratio between the highest and the lowest value of the Jacobian determinant inside a mesh element. It is equal to 1 in an element where the angles between all line segments are identical [12]. To evaluate mesh quality, the average of the scaled Jacobian of each mesh cell will be used as the quality score.

Three meshing programs where selected for quality comparison. The programs where selected on two criteria: ease of integration into Python and compatibility (e.g. does the software accept the correct file format). This resulted in three possible algorithms: Mmg, ACVD and TriMesh.

Mmg is an open source remeshing software [13]. To make the software compatible with python, the pymmg python wrapper [14] was used. ACVD is a Voronoi-Distribution based remeshing algorithm [15], for which exists a Python port named pyACVD [16]. Finally, TriMesh is a Python native library for working with triangular surface meshes [17]. All of these programs where used to generate a remeshed version of the input wall geometry. The settings of the algorithms are chosen to ensure that each resulting mesh has a similar number of nodes. The settings can be found in Table 2.2.

The results of the analysis are shown in Table 2.3. The best performing meshing algorithm in terms of quality is Mmg. In the similarity metric TriMesh performs the best. However, this comes at a great cost in terms of quality. The cause for these divergent results is the method that TriMesh uses to refine the mesh: the method does not re-evaluate the entire geometry but subdivides the existing triangles into smaller ones. This leaves all surfaces exactly the same, but fails to improve low quality areas in the original mesh. In Figure 2.8 on the right side it can be seen that the similarity is perfect everywhere on the mesh. ACVD rivals Mmg in terms of quality, but the similarity is almost ten times lower for ACVD. In a large part this is caused by ACVD shifting the boundary edges. In Figure 2.8 on the left side it can be seen that, especially around the inlet edge, the similarity drops drastically. Based on these findings Mmg is chosen as the preferred surface meshing tool.

Table 2.2: *Settings used for mesh generation*

Software	Setting
ACVD	Subdivide = 4; Cluster = 7,5*num_of_nodes
Mmg	-hausd 0.1 -hsiz 1
TriMesh	subdivide_to_size(1,97)

Table 2.3: *Results of the quality assessment*

Software	Mean similarity	Mean quality
ACVD	0,0333	0,8723
Mmg	0,0053	0,9119
TriMesh	0,0000	0,6364

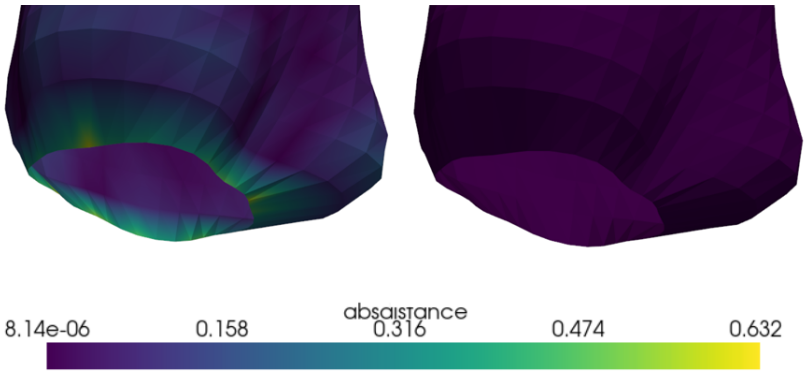


Figure 2.8: *Visualisations of similarity of the acvd mesh (L) and trimesh (R)*

2.4. Manual vs Python simulations

The first FEBio simulations are done as a proof of concept both manually and with the python script. The finalized 3D mesh was used as an input for the manual simulations, and the boundary conditions and material values were added manually. These are done with a low velocity of 0.2m/s, parabolic constant inflow profile and short end time of t = 0.8seconds, for easy and quick simulations. Figure 2.9 and figure 2.10 show that both methods give the same result, proving the python script works the same as a manual FEBio simulation.

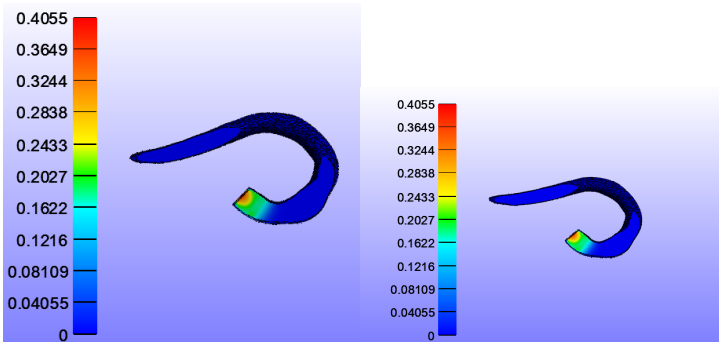


Figure 2.9: *Geometry 1: Manual on the left vs Python simulation on the right at the same time point*

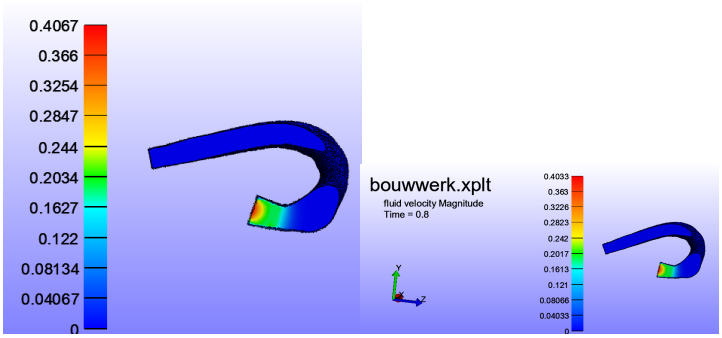


Figure 2.10: *Geometry 4: Manual vs Python simulation at the same timepoint*

However, after manually importing the geometries 2 and 3 in FEBio, the software does not recognize the output surfaces, as seen in figure 2.11 and 2.12. The corners are too curved to be seen as separate surfaces. Because of this the zero pressure boundary conditions could not in these manual simulations.

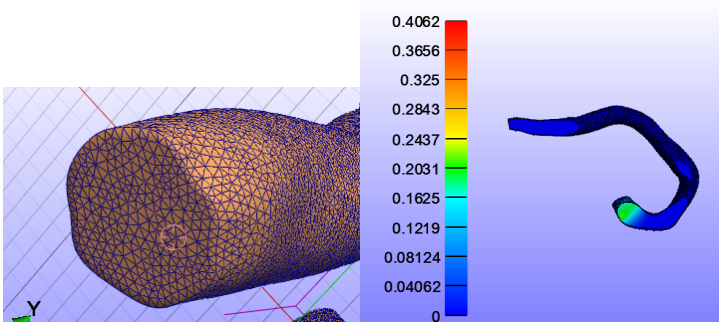


Figure 2.11: *Geometry 2: No recognition of the ouput surface in FEBio for manual simulation on the left vs Python simulation on the right*

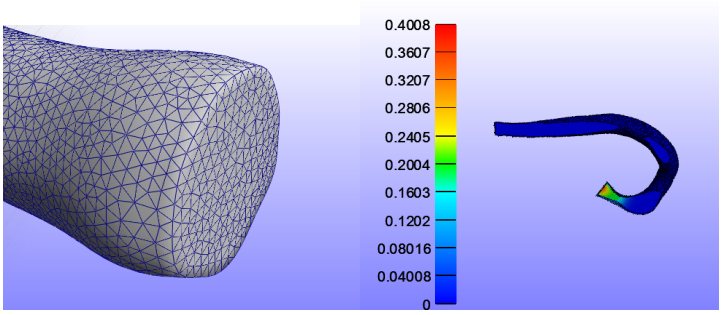


Figure 2.12: *Geometry 3: No recognition of the ouput surface in FEBio for manual simulation vs Python simulation*

The automated workflow, however, does recognise all surfaces. Swapping the inlet to outlet and vice versa, as seen in figure 2.13, gives a visual proof that the boundary conditions can be added to a surface that FEBio normally would not recognise. This is because the minimum angle at which FEBio detects a surface cannot be changed, but it can be changed to the required value in the python script.

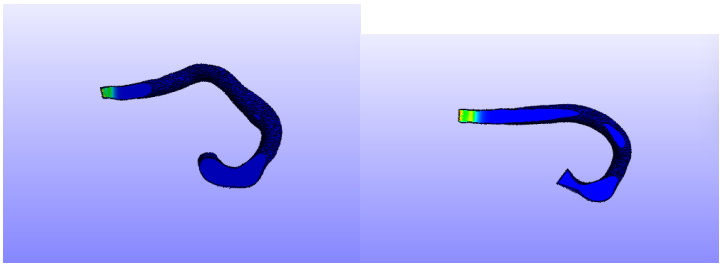


Figure 2.13: *Geometry 2 and 3 with swapped inlet and outlets to prove surface identification*

Lastly, Now that it has been shown that the python script works, a full simulation has been done. This time the velocity is increased to a more realistic value of 1 m/s.

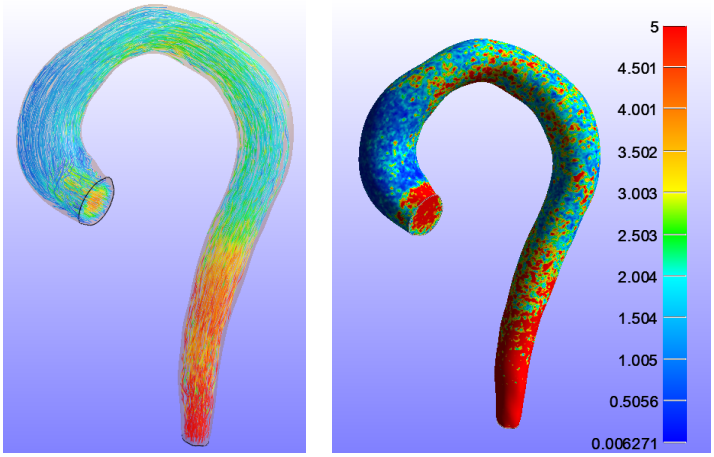


Figure 2.14: *Full CFD simulation with streamlines and absolute velocity (Left) and fluid acceleration(Right)*

Figure 2.14 clearly shows the parabolic velocity profile on the aortic inlet. The fluid goes slower in the first part of the aorta, where the tube is wide, and then accelerates further when the tube becomes more narrow. It takes about 0.4 seconds for the fluid to travel through the whole geometry, which intuitively seems correct. A rough calculation by hand gives $A_{inlet} = 0.00082m^2$, $A_{outlet} = 0.000157m^2$, $v_1 = 1m/s$
$v_2 = v_1 * A_1/A_2 = 5.2m/s$. taking the average velocity, with a pipe roughly 20 cm long gives $t = v_{average} * 0.2m = 0.6s$, which is the same order of magnitude as our simulation.

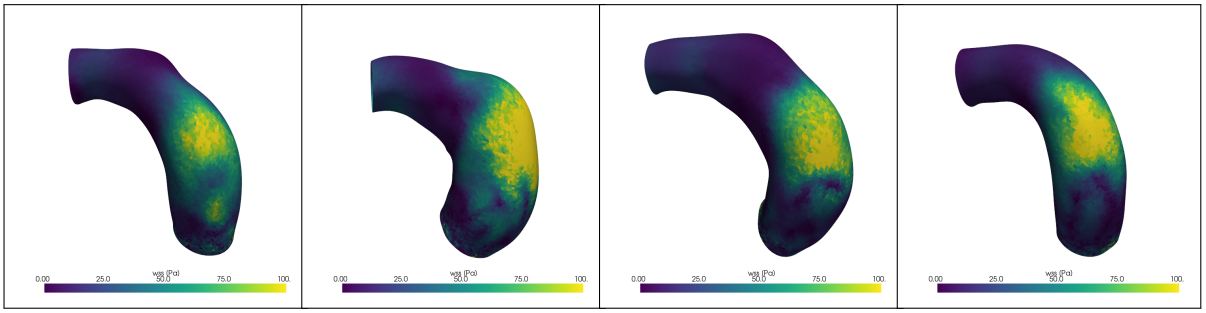
3

Results

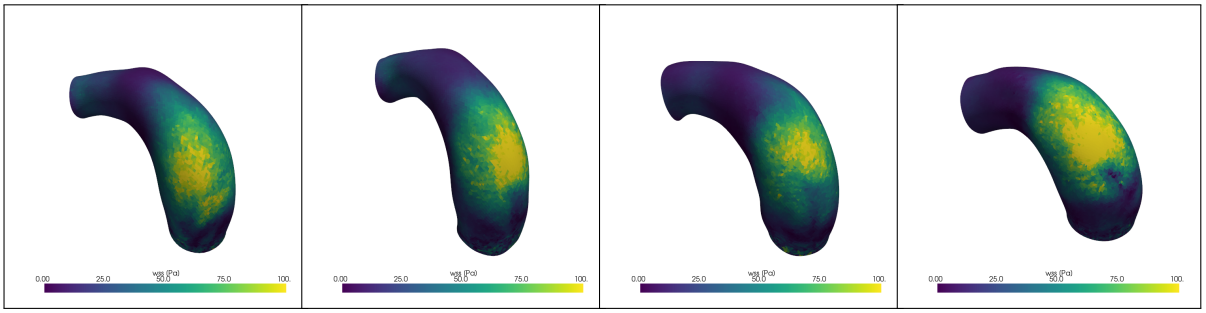
Included here are visual results showing the outcome of ten simulations (Figures 3.1) as well as the WSS data (Table 3.1).

Table 3.1: *WSS results*

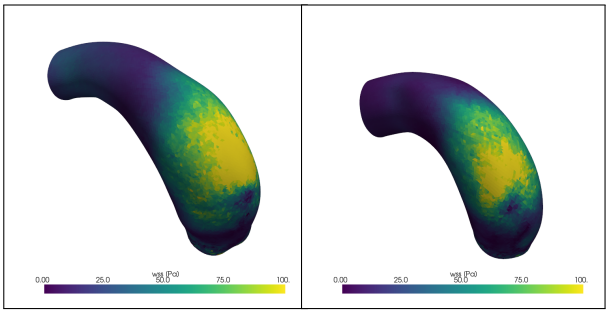
	Case								
Percentile/WSS [Pa]	221	326	394	430	436	442	468	480	552
99	91.25	126.99	98.79	103.19	95.83	102.83	92.18	114.74	124.64
98	83.20	117.53	89.77	95.03	88.79	91.33	83.36	103.15	109.51
96	70.85	104.73	79.07	79.92	77.47	76.25	72.08	89.34	91.74
94	62.64	94.53	70.16	69.17	70.18	68.20	64.57	80.52	80.62
92	56.44	86.58	61.93	59.51	63.45	62.54	59.30	73.73	72.84
90	51.73	79.13	54.22	51.91	57.45	56.94	54.23	67.96	65.89



(a) *Case 0221* **(b)** *Case 0326* **(c)** *Case 0394* **(d)** *Case 0430*



(e) *Case 0436* **(f)** *Case 0442* **(g)** *Case 0468* **(h)** *Case 0480*



(i) *Case 0552* **(j)** *Case 0578*

Figure 3.1: *WSS plots for selected cases*

Bibliography

[1] Romero, P., Lozano, M., Martínez-Gil, F., Serra, D., Sebastián, R., Lamata, P., and García-Fernández, I., "Clinically-Driven Virtual Patient Cohorts Generation: an Application to AoRta," *Frontiers in physiology*, Vol. 12, 2021. doi:10.3389/fphys.2021.713118, URL https://doi.org/10.3389/fphys.2021.713118.

[2] Saitta, S., Maga, L., Armour, C. H., Votta, E., O'Regan, D. P., Salmasi, M. Y., Athanasiou, T., Weinsaft, J. W., Xu, X. Y., Pirola, S., and Redaelli, A., "Data-driven generation of 4D velocity profiles in the aneurysmal ascending aorta," *Computer methods and programs in biomedicine*, Vol. 233, 2023, p. 107468. doi:10.1016/j.cmpb.2023.107468, URL https://doi.org/10.1016/j.cmpb.2023.107468.

[3] Telea, A., and Vilanova, A., *A Robust Level-Set Algorithm for Centerline Extraction*, University of Groningen, Johann Bernoulli Institute for Mathematics and Computer Science, 2003. Relation: http://www.rug.nl/informatica/organisatie/overorganisatie/iwi Rights: University of Groningen. Research Institute for Mathematics and Computing Science (IWI).

[4] Younas, S., and Figley, C. R., "Development, implementation and validation of an automatic centerline extraction algorithm for complex 3D objects," *Journal of medical and biological engineering*, Vol. 39, No. 2, 2018, pp. 184–204. doi:10.1007/s40846-018-0402-1, URL https://doi.org/10.1007/s40846-018-0402-1.

[5] "vmtk - the Vascular Modelling Toolkit," , ???? URL http://www.vmtk.org/index.html.

[6] Piccinelli, M., Veneziani, A., Steinman, D., Remuzzi, A., and Antiga, L., "A framework for Geometric Analysis of Vascular Structures: Application to cerebral Aneurysms," *IEEE transactions on medical imaging*, Vol. 28, No. 8, 2009, pp. 1141–1155. doi:10.1109/tmi.2009.2021652, URL https://doi.org/10.1109/tmi.2009.2021652.

[7] Izzo, R., Steinman, D., Manini, S., and Antiga, L., "The vascular modeling toolkit: a python library for the analysis of tubular structures in medical images," *Journal of Open Source Software*, Vol. 3, No. 25, 2018, p. 745.

[8] Csippa, B., Sándor, L., and Paál, G., "Decomposition of Velocity Field Along a Centerline Curve Using Frenet-Frames: Application to Arterial Blood Flow Simulations," *Periodica Polytechnica Mechanical Engineering*, Vol. 65, No. 4, 2021, pp. 374–384.

[9] Yahya-Zoubir, B., Hamami, L., Saadaoui, L., and Ouared, R., "Automatic 3D Mesh-Based Centerline Extraction from a Tubular Geometry Form," *Information Technology And Control*, Vol. 45, 2016. doi:10.5755/j01.itc.45.2.12162.

[10] Schaefer, B. M., Lewin, M. B., Stout, K. K., Gill, E., Prueitt, A., Byers, P. H., and Otto, C. M., "The bicuspid aortic valve: an integrated phenotypic classification of leaflet morphology and aortic root shape," *Heart*, Vol. 94, No. 12, 2008, pp. 1634–1638.

[11] Sorgente, T., Biasotti, S., Manzini, G., and Spagnuolo, M., "A survey of Indicators for Mesh Quality Assessment," *Computer graphics forum*, Vol. 42, No. 2, 2023, pp. 461–483. doi:10.1111/cgf.14779, URL https://doi.org/10.1111/cgf.14779.

[12] Engvall, L., and Evans, J. A., "Mesh quality metrics for isogeometric Bernstein–Bézier discretizations," *Computer Methods in Applied Mechanics and Engineering*, Vol. 371, 2020, p. 113305. doi:https://doi.org/10.1016/j.cma.2020.113305, URL https://www.sciencedirect.com/science/article/pii/S0045782520304904.

[13] MmgTools, "GitHub - MmgTools/mmg: open source software for bidimensional and tridimensional remeshing," , ???? URL https://github.com/MmgTools/mmg.

[14] "pymmg V1.0.0," , 2024. URL https://pypi.org/project/pymmg/.

[15] Valette, "GitHub - valette/ACVD: Fast simplification of 3D surface meshes," , 2024. URL https://github.com/valette/ACVD?tab=readme-ov-file.

[16] Pyvista, "GitHub - pyvista/pyacvd: Python implementation of surface mesh resampling algorithm ACVD," , 2024. URL https://github.com/pyvista/pyacvd.

[17] Dawson-Haggerty et al., "trimesh," , 2024. URL https://trimesh.org/.