Resum

Aquest projecte de final de carrera s'emmarca dins un estudi de recerca d'un estudiant de doctorat, Julien Egger de la “Swiss Federal Institute of Technology of Zürich (ETH)”.

Té com a finalitat el desenvolupament d’un programa per optimitzar el posicionament virtual d'un desviador de flux (FD) dins d'una artèria en el cas d'un aneurisma, definint un aneurisma com la dilatació de l’artèria causada per la degradació de la paret. També servirà de base per després fer diferents simulacions de flux i així tenir una anàlisi de sensibilitat sobre les diferents característiques del FD, una vegada implementat en un pacient.

Concretament, saber com el FD té influència sobre l'artèria, en el flux sanguini, ja que l'objectiu d'utilitzar un FD és canviar la direcció del flux per tal d’evitar que entri a l'interior del sac, fet que portaria a un augment del mateix podent-ne provocar l’explosió.

A més a més, trobar una solució per resoldre els possibles problemes en cas que no funcioni i per donar més informació als cirurgians sobre el procediment de implementació. Per exemple, pot servir alhora de comparar les diferents opcions en cas d'una bifurcació, en quina direcció ha d'anar el FD? Per provar nous enfocaments pràctics (dos FD un dins l'altre), etc.

Per tant, donada les dades d’un volum en tres dimensions obtingut a partir d’una CT d'un pacient, el mètode desenvolupat en aquest treball té per objectiu el posicionament d'un FD virtual dins de l’artèria. El procediment emprat es pot dividir en els següents subpassos:

- Obtenir una representació en 3D de la paret de l'artèria per mitjà de la segmentació.
- Trobar la representació matemàtica de la línia central de l'artèria per utilitzar-la com a base per a la descripció geomètrica del desviador.
- Donades les característiques del FD considerat, adaptar la geometria al llarg de la línea central per tal que encaixi amb l’artèria.
- Exportar i guardar la representació volumètrica del FD en un format determinat, compatible amb els productes comercials utilitzats per a mallats volumètrics i simulació de flux.

Per acabar dir que comparacions qualtitatives del posicionament virtual de FD obtingut a partir d’aquest mètode i el posicionament de FD obtingut a partir de dades reals de pacients són presentats en aquest projecte.

Abstract

This bachelor thesis (PFC) is framed in a research study from a PhD student, Julien Egger from the Swiss Federal Institute of Technology of Zürich (ETH).

It's aim is to develop a program to optimize the position of a virtual flow diverter inside an artery in case of an aneurysm, defining an aneurysm as a dilatation of an artery caused by the degeneration of the wall.

This is the base for making different flow simulations to have a sensitivity analysis on the different characteristics of the flow diverter (FD) once it is implemented into a patient.

Concretely to know how the flow diverter has influence inside the artery, in the blood flow. The goal of using a FD is to change the flow direction to avoid it entering inside the sack leading to an augment of it that may cause bursting.

Also to find a solution to solve the possible problems in case it doesn't work and to give more information to the surgeons about the implementation procedure. For example to compare different options in case of a bifurcation, in which direction has to go the flow diverter? And to test new approaches virtually (two FD inside each other), etc.

Given a volume of 3 dimensional data obtained from the CT Scan of a patient, the method developed in this work aims at obtaining the position of a virtual flow diverter inside an artery.

This procedure can be divided into the following substeps:

- Obtain a 3 dimensional representation of the wall of the artery trough segmentation.
- To find the mathematical representation of the centreline of the artery to serve as a backbone for the geometrical description of the flow diverter.
- Given the characteristics of the considered flow diverter, adapt its geometry along the centreline to fit into the vessel.
- Export and save the volumetric representation of the flow diverter into a specified format, compatible with commercial products used for volumetric meshing and flow simulations.

As a conclusion, qualitative comparisons of the position of virtual flow diverters obtained with this method and the position of flow diverters obtained from real patient data will be presented.

Summary

RESUM 1

ABSTRACT 2

SUMMARY 4

FIGURE INDEX 7

TABLE INDEX 11

1. GLOSSARY 13

2. SOFTWARE PACKAGES AND LIBRARIES 14

3. SCOPE AND OBJECTIVES 16

3.1 Motivation 16

3.2 Objectives 17

3.3 Planning 19

4. BACKGROUND 21

4.1 Medical background 21

4.2 Technological background 29

5. CONTRIBUTION 35

5.1 Methodology 35

5.2 Procedure followed 35

5.2.1 Three-dimensional Vessel Representation 36

5.2.1.1 Segmentation 36

5.2.1.2 Handling of the sections 43

5.2.1.3 Mesh generation 45

5.2.2 Generation of Flow Diverter 48

5.2.3 Output 77

6. RESULTS 79

7. COST OF THE PROJECT 87

8. SOCIAL & ENVIRONMENTAL IMPACT 89

CONCLUSIONS 90

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

ACKNOWLEDGEMENTS 91

BIBLIOGRAPHY 92

8.1 Bibliographic references 92

8.2 Complementary bibliography 93

ANNEXES 95

ANNEX A: MATLAB CODES 95

ANNEX B. Produce1DStraightStent.m 195

ANNEX C. Code to test the results 197

ANNEX D. Distance Between Point and Triangle in 3D 202

Figure Index

Figure 2.1: Initial segmented data……..18

Figure 2.2: Virtual FD implemented……...…….18

Figure 3.1: Circle of Willis…….21

Figure 3.2: Example of types of aneurysms……..22

Figure 3.3: Clipping treatment…….23

Figure 3.4: Example of the skull incision and the different parts…….24

Figure 3.5: Insertion of the clip ……...24

Figure 3.6: Procedure for inserting the coils…..…….25

Figure 3.7: Implementation of the stent in an aortic aneurysm……...26

Figure 3.8: Example of a flow diverter & stent……...26

Figure 3.9: Virtual endoscopy process……...…….29

Figure 3.10: A medial axis in 2D (a and b) and a medial surface in 3D (c) and a few examples of inscribed discs (2D) and a ball (3D).……..……..31

Figure 3.11: A medial surface in 3D……..…….31

Figure 3.12: Example of the thinning algorithm…….32

Figure 3.13: Example using the distance field algorithm…….32

Figure 3.14: Example of a VORNOI Diagram……...33

Figure 3.15: Reeb graph example……..34

Figure 4.1: Summary of the first three steps…….36

Figure 4.2: Initial screen when the users open the program……...38

Figure 4.3: The data loaded; see the images in the 2D viewer……..39

Figure 4.4: Slice number 349 from the CT……39

Figure 4.5: A slice of the CT showing the part selected using the threshold tool……40

Figure 4.6: A 3D view during the segmentation……41

Figure 4.7: A zoom to see better the voxels selected...……...42

Figure 4.8: Output after the segmentation using Slicer3……43

Figure 4.9: Box used to cut the proximal section...……..…….…44

Figure 4.10: Final result after cutting the surfaces……44

Figure 4.11: Variables to change I……..……..……..46

Figure 4.12: Variables to change II……..……...46

Figure 4.13: The surface mesh used for the next steps…...……...47

Figure 4.14: One helix……..48

Figure 4.15: Straight FD……...49

Figure 4.16: One triangle-element……..51

Figure 4.17: Vessel after plotting the STL file with MATLAB……..52

Figure 4.18: Selecting the first cover…….……53

Figure 4.19: Region selected…….……..54

Figure 4.20: Selecting one element…….……...54

Figure 4.21: Box dialogue (1)……...……...……...54

Figure 4.22: Example selecting the second cover…….…….....55

Figure 4.23: Example of the BB……...……...56

Figure 4.24: CM in the region selected with the BB……57

Figure 4.25: Box dialogue (2)……57

Figure 4.26: Select a new region……58

Figure 4.27: The new CM……58

Figure 4.28: Example of the points that are inside the BB……58

Figure 4.29: Partial CM……59

Figure 4.30: Example during the rasterization process……60

Figure 4.31: Example of checking a point……61

Figure 4.32 Example of checking if it cross……62

Figure 4.33 Example of checking the direction……62

Figure 4.34: The two green arrows show the direction of the interpolation line between the two points……63

Figure 4.35: Cubic spline interpolation line between two points……63

Figure 4.36: CAS and a zoom……65

Figure 4.37: All the radius represented……66

Figure 4.38: Example of the plane π and the triangles that intersect the plane……67

Figure 4.39: Radius plotted as a circle……68

Figure 4.40: A test with different CM……69

Figure 4.41: Vessel radius previous to the optimization process……70

Figure 4.42: Interpolation radius in the neck of the aneurysm……72

Figure 4.43: Centreline before and after the optimisation process……73

Figure 4.44: One coil “going forward” and one “going backwards”……75

Figure 4.45: Virtual FD inserted inside the vessel……76

Figure 4.46: Zoom of the FD……77

Figure 4.47: Virtual flow diverter after modelling the radius and put it around the vessel…...77

Figure 4.48: The vessel in blue and the flow diverter in red……78

Figure 5.1: One slice of the CT where we can see the FD……..79

Figure 5.2: The two vessels from we extract the envelops to compare……80

Figure 5.3: Distance between the two centrelines……80

Figure 5.4: Two different envelops……..81

Figure 5.5: Plotting the value of the radius of the CM’s...…….....…82

Figure 5.6: Difference between the different radius MinRad-R and MaxRad-R…….....83

Figure 5.7: Radius Ratio (1).……84

Figure 5.8: Radius Ratio (2).……...85

Figure 5.9: Distortion of the CT……..…….....85

Figure 5.10: Fig. 5.10 Summary of the different results obtained before.……86

Table Index

Table 2.1: Gantt Diagram..……..……...20

Table 3.1: Advantages and disadvantages of a virtual colonscopy……..…….29

Table 4.1: CT information……..……..39

Table 4.2: Threshold range……..……...40

Table 4.3: Surface characteristics…….…..47

Table 4.4: STL data……..……49

1. Glossary

CM → Centre of Mass

CAS → Coordinates Axis System

FD → Flow Diverter

ROI → Region Of Interest

CT → Computer Tomography

BB → Bounding Box

ICEM CFD → ANSYS ICEM CFD meshing software

MRI → Magnetic Resonance Imaging

HFS → Head First-Supine, DICOM data orientation

F → Faces

V → Vertices

N → Normal

DICOM → Digital Imaging and Communication in Medicine.

2. Software packages and libraries

SLICER3.4 → web [A free, open source software package for visualization and image analysis]*, *[http://www.slicer.org/, 17/04/2011]

PARAVIEW → web [is an open-source, multi-platform data analysis and visualization application. ParaView users can quickly build visualizations to analyse their data using qualitative and quantitative techniques. The data exploration can be done interactively in 3D or programmatically using ParaView's batch processing capabilities.]*, *[http://www.paraview.org/, 05/05/2011]

MATLAB → web [short for Matrix Laboratory, "matrix laboratory" is a mathematical software that offers integrated development environment (IDE) with an own programming language (M)]*, *[http://www.mathworks.com/, 21/02/2011]

STL format → web [STL files describe only the surface geometry of a three dimensional object without any representation of colour, texture or other common CAD model attributes. The STL format specifies both ASCII and binary representations.]*, *[http://www.ennex.com/~fabbers/StL.asp, 22/02/2011]

3. Scope and Objectives

3.1 Motivation

An aneurysm is an abnormal widening of a piece of an artery. It appears when the artery weakens over time and the wall starts to expand. This usually occurs at the junctions of the large arteries at the base of the brain, in the polygon of Willis or in the abdominal aorta, concretely in the descending aorta.

Three parameters are important to define an aneurysm, the diameter of the neck, the size of the dome and the depth.

There are no known ways to prevent the formation of a cerebral aneurysm (the ones that are been studied in this thesis) and most of the aneurysms go unnoticed until they break or they are detected by brain imaging when the doctor is checking another illness.

Even if numerous cases of aneurysms are never discovered and remain in a stable harmless state without treatment, those that are discovered still have to be investigated and maybe treated because of the major risk that they represent upon rupture. Its rupture could cause the death of the patient.

In case of an aneurysm in the brain, the aneurysm can break and bleed into the brain causing serious complications such as haemorrhagic stroke, permanent nerve damage, or both. The rupture also may cause a subarachnoid haemorrhage (SAH), bleeding into the space between the skull and the brain.

In the aortic artery, the aneurysms have also a big impact because the break of the dome frequently causes the death of the patient. The most common aneurysms are the ones in the abdominal aorta. “Men have more probabilities than women to have an aneurysm” **[8]. “**It is estimated that up to one in 15 people in the United States will develop a brain aneurysm during their lifetime.” **[9]**

Surgeons have different methods to treat patients like surgical intervention, which is a high invasive technique (clipping) or endovascular techniques (coiling or introduction of a flow diverter).

Patients with an aneurysm may suffer deficits during a period of time as a result of the rupture of it or after the treatment due to the kind of intervention that has risks.

Although these methods are now well established and have been proven very efficient in practice, some questions are still unanswered. For example:

- Once the surgeon has found the aneurysm and implements a flow diverter (FD), when using a minim invasive technique, there is no possibility to be sure that this will solve the problem, in some cases the aneurysm disappears, but in others the dome breaks after few days or directly the treatment doesn't make any effect into the aneurysm. Why do some aneurysms break after placement of a flow diverter ? Is it related to thrombosis?
- In case of a bifurcation, where the surgeon has to place the flow diverter ?
- To give grounds to choose which FD depending on its porosity.
- The surgeon has to introduce one or two flow diverters, one inside the other?

The final objective of the project, in which this work is framed, is to analyse the blood flow after placing a containment device into the artery, called flow diverter (FD), defining it as a device that reduces the velocity of the blood flow inside the aneurysm.

A potential candidate method to investigate these questions is the flow simulations. For making these simulations a model for flow is needed (existing) and also a model for blood coagulation, part that has to be investigated and this is one of the most important aim of the all project.

The project presented here will contribute to the bigger study conducted mainly the Computer Vision Department of the Swiss Federal Institute of Technology in Zürich (ETH) and the Hirslanden Clinic in Zürich.

3.2 Objectives

The present work address the problematic of obtaining a realistic model of a flow diverter, with predefined specifications placed into a cerebral blood vessel, following the next substeps:

- To segment the vessel from the data obtained from a patient by using medical images. Specifically, given a set of images from a Computer Tomography (CT). A CT is a medical imaging technique non-destructive that produces 2D and 3D cross-sectional

images of a patient from flat X-Ray images. It gives characteristics of the internal structure of the patient such as dimensions, shape, internal defects and density.

- To extract a 3D mesh of the segmented part of the vessel in such a way that it can be exported to MATLAB program.
- To find a mathematical strategy to extract the midline of the vessel.
- To develop the program to virtually arrange any flow diverter along the centreline.
- To find quantitative means to compare the position of the virtual stent with the position of the real FD.
- To make recommendations for the next improvements of the procedure.

This is illustrated in Figure 2.1, where the image shows the segmented data and in Figure 2.2 the virtual flow diverter (FD) after the execution of the program (the FD is the pink one and the vessel with the aneurysm the blue one). These two figures help to understand where the process starts and where it finishes.

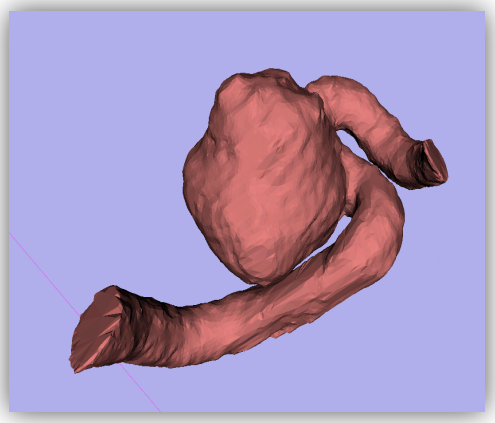


Fig.2.1 Initial segmented data

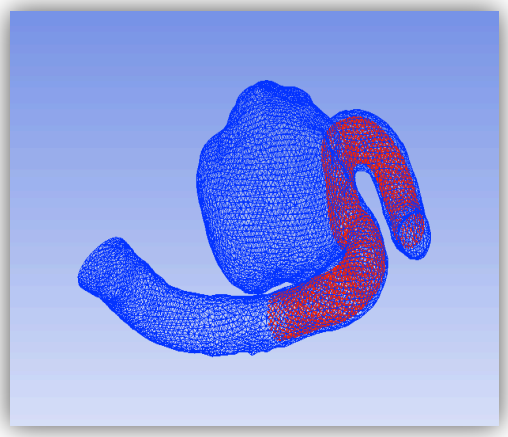


Fig. 2.2 Virtual FD implemented

3.3 Planning

Gantt Diagram of the time spent during my stage in Zurich doing my “PFC” and the time I spent writing the report in Barcelona.

T1. Analysis and formation

T1.1 Biomedical background: Increase my knowledge in this field. Learn about aneurysm and its effects. Learn about the possible medical interventions.

T1.2 Virtual endoscopy background: What is a virtual endoscopy and which are the different methods?

T1.3 Design of the methodology: definition of the successive steps

T2 Analysis and development of the code

T2.1 To obtain the centre of mass (CM)

T2.2 To select partial segments: to select only the relevant part of the vessel

T2.3 To produce the centreline: parametric interpolation spline to connect all the CM’s

T3 Search and produce the Output

T3.1 To find a way to produce the virtual FD inside the vessel: Register Nominal Frenet Coordinate Axes, Get circumferential vessel radius function (minimum and maximum radius), FD envelope, without previous optimization

T3.2 To produce the virtual FD: produce the coils

T3.3 To produce STL File with the virtual FD: creation of a data to work

T4 Formation to use a program to segment and segmentation

T4.1 Methodology to segment: increase my knowledge in the segmentation field

T4.2 Segmentation: take the part from the CT that has the aneurysm

T5 Compare the results between a real pre and post operation data of a patient

T6 Optimization of the process and time efficiency

T6.1 Centreline: to optimize the creation of the centreline in the neck of the aneurysm

T6.2 Curvature: to modify the virtual FD depending the curvature of the vessel

T6.3 CM in the covers: different way to find the CM in the covers

T7 Writing: to write the thesis

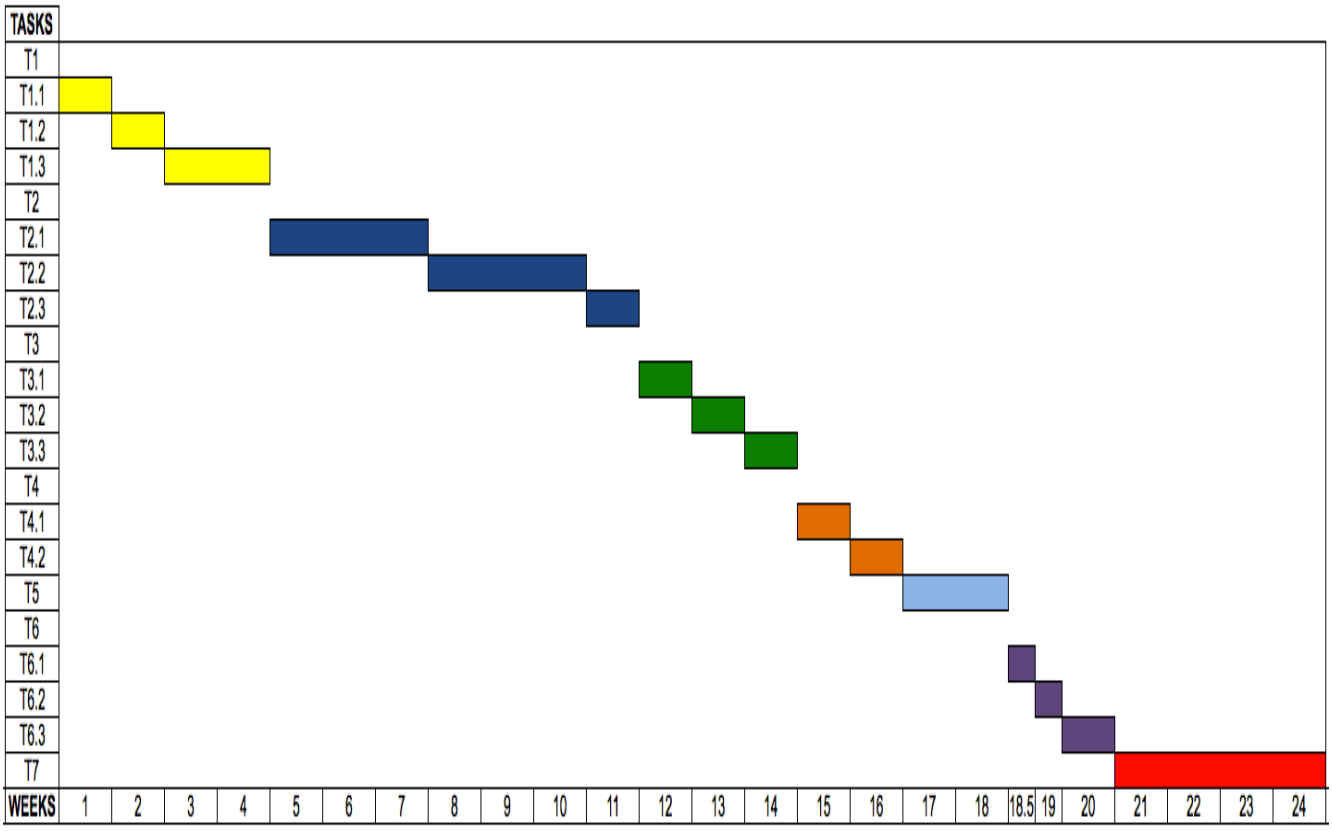


Table 2.1 Gantt Diagram

4. Background

4.1 Medical background

4.1.1 Pathologies

Cerebrovascular accidents are the third leading cause of death after heart attack and cancer. It consists on the lack of oxygen in the brain **[7]**. There are two types of strokes accidents:

1. Ischemic stroke: it is based on the decrease of cerebral blood flow, under a threshold is dangerous.
2. Subarachnoid haemorrhage (SAH): in this case 80% of deaths are caused by the rupture of the aneurysm (dome of the aneurysm), causing an intracranial bleeding. This aneurysm commonly appear in the circle of Willis localized in the neck of the body.

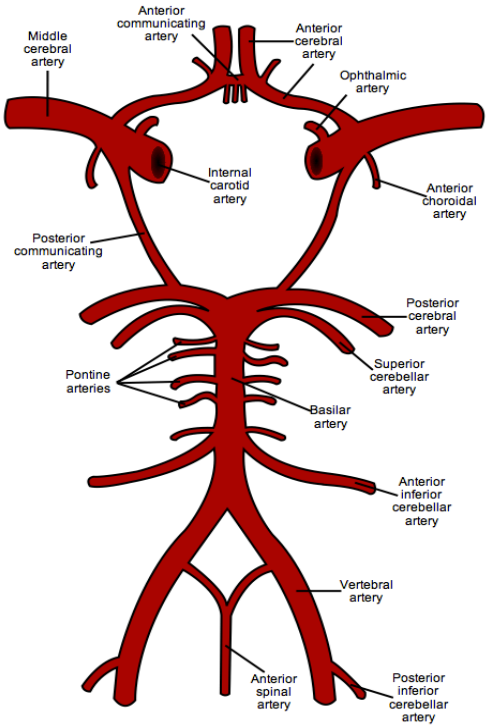


Fig.3.1 Circle of Willis (from http://upload.wikimedia.org/wikipedia/commons/2/2e/Circle_of_Willis_en.svg)

An aneurysm is a localized dilatation of an artery or a vein (arterial aneurysms are much more common) caused by the degeneration of the walls. There are different types of aneurysm classified by their morphology (macroscopic shape and size); there are the “sacular” aneurysms or the fusiform ones.

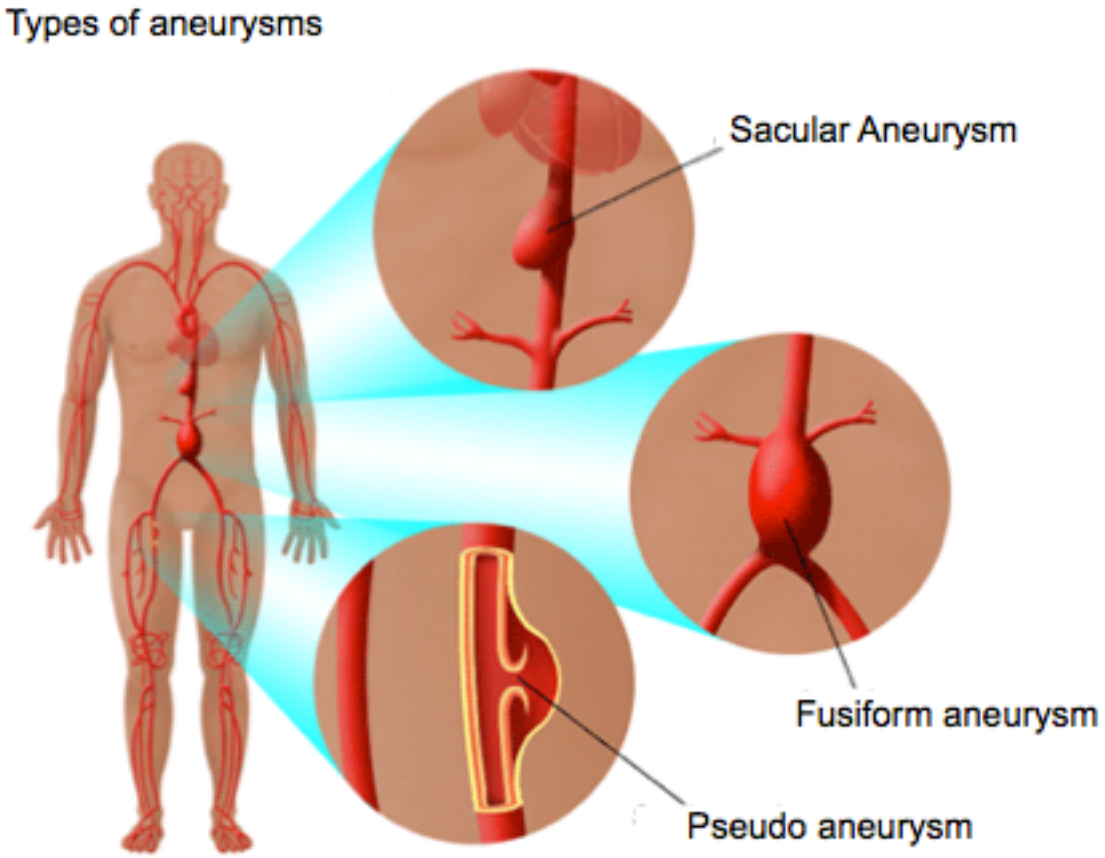


Fig.3.2 Example of types of aneurysms. Changed to English (original image from http://www.reshealth.org/sub_esp/yourhealth/healthinfo/default.cfm?pageID=P08287)

The “sacular” aneurysms are spherical and involve a portion of the vessel wall (5-20 cm of diameter).

On fusiform aneurysms involve large portions of the vessel. There are more likely to be found on the ascending and transverse aortic arch, the abdominal aorta or on the iliac arteries. The diameter can extend to 20cm.

A pseudo-aneurysms could appear in any artery and its diameter increases over time due to the arterial pressure.

There are different causes that could generate an aneurysm, but the most common are:

- Genetic causes
- Natural causes as the aging of the artery
- Environmental factors as tobacco, high cholesterol, etc.

4.1.2 Treatments of aneurysms

There are three types of treatments:

- Clipping
- Coiling
- Implement a flow diverter

4.1.2.1 Clipping

Clipping is a surgical intervention that consists in closing the base of the aneurysm with a clip. The handling of the aneurysm, especially the dome, can cause rupture.

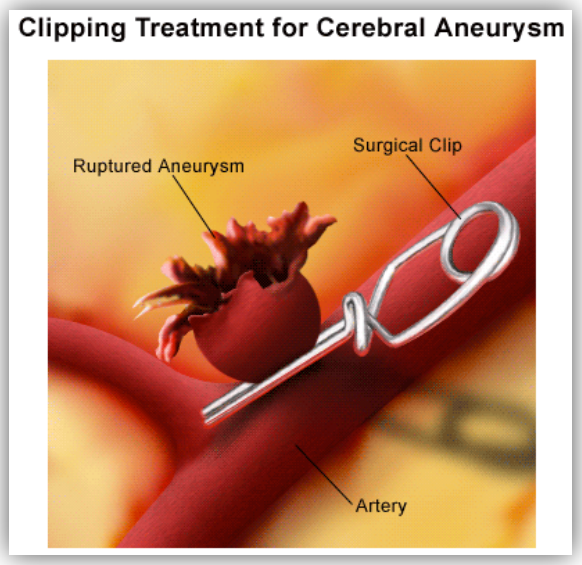


Fig.3.3 Clipping treatment (from http://www.jeffersonhospital.org/diseases-conditions/cerebral-aneurysm.aspx?disease=4581a9cd-6b07-43e0-a729-aa6994d4b750)

- First of all the surgeon starts making an incision in the skin and then he/she makes a small hole in the skull with a drill. Thanks of this hole, the surgeon can introduce a special saw to cut a piece of the skull bone. Then it is removed.

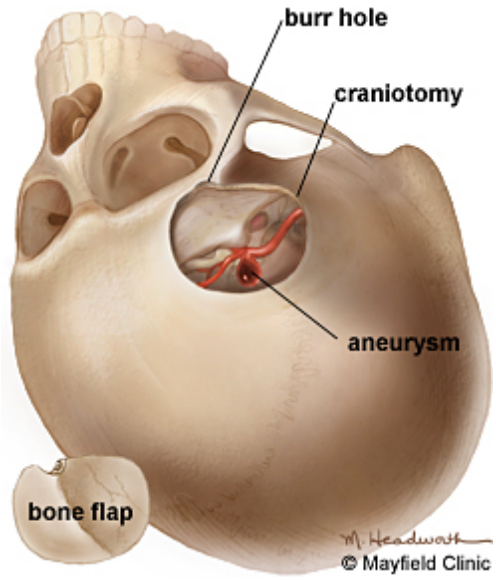


Fig.3.4 Example of the skull incision and the different parts (from http://www.mayfieldclinic.com/PE-Clipping.htm)

- Once the piece of bone is lifted the surgeon carefully finds the artery and follows it up to the aneurysm and inserts the clip.

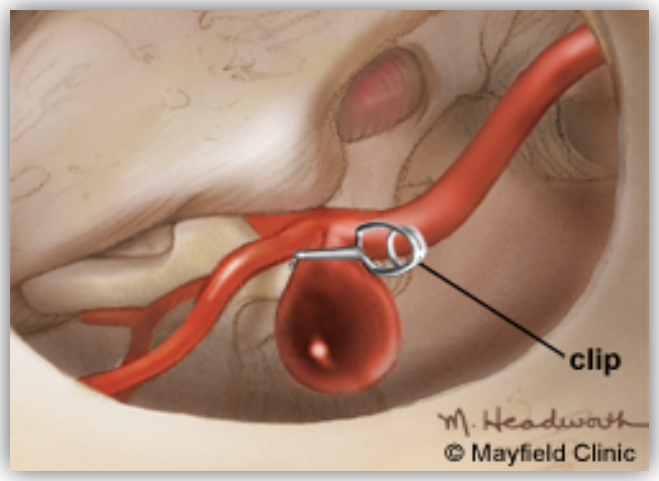


Fig. 3.5 Insertion of the clip (from http://www.mayfieldclinic.com/PE-Clipping.htm)

During and after the intervention process the patient has to be aware certain consequences and implications, inherent to the procedure. The most important are cited below:

- Losing of the pressure brain when the surgeon opens the skull could cause some injuries.
- The procedure needs the use of general anaesthesia.
- Vasospasm, stroke, seizure, bleeding, and an imperfectly placed clip, which may not completely block off the aneurysm or blocks a normal artery unintentionally.
- Patients could have deficits as a result of the rupture of the aneurysm or after the treatment during a short or long period of time as it is explained before.

4.1.2.2 Endovascular treatments

The endovascular treatments that tend to replace the heavy invasive open surgical procedures are:

Coiling

In the case of brain aneurysms, the procedure used is: to pass a catheter from the femoral artery in the groin to the aneurysm in the brain, going through the aorta, continuing into the brain arteries where the aneurysm is located. Then the surgeon introduces platinum coils in the sack. The coils are packed into the aneurysm causing the blood clotting within the aneurysm. The goal of introducing coils and the blood clotting is to fill the aneurysm closing it off and preventing blood continues entering into the sack. Then a Stent that is a small cylindrical mesh tube is introduced via catheter and covers the neck or entrance of the aneurysm, holding the coils in place.

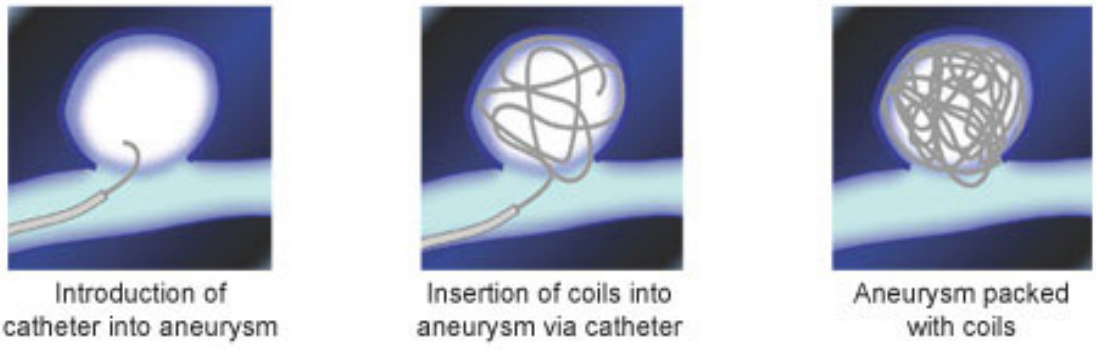


Fig.3.6 Procedure for inserting the coils (from http://www.taafonline.org/ba_treatment.html)

In Figure 3.6 the introduction of the coils inside of the aneurysm via a catheter is shown and in Figure 3.7 how the stent is implemented:

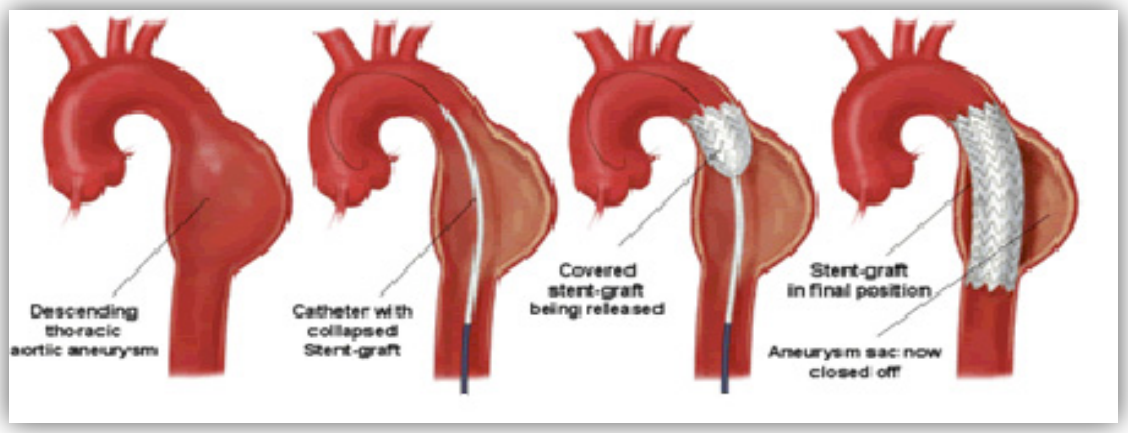


Fig.3.7 Implementation of the stent in an aortic aneurysm (from http://www.massgeneral.org/heartcenter/services/procedure.aspx?id=2182)

Flow diverter: The implementation of a flow diverter follows a similar procedure as for the placement of a stent. The only changes are the shape and its function: the flow diverter acts directly as the flow of blood, reducing its velocity inside the aneurysm whereas the stent is mainly responsible for keeping the coils inside the aneurysm. See the Figure 3.8 to notice the difference.

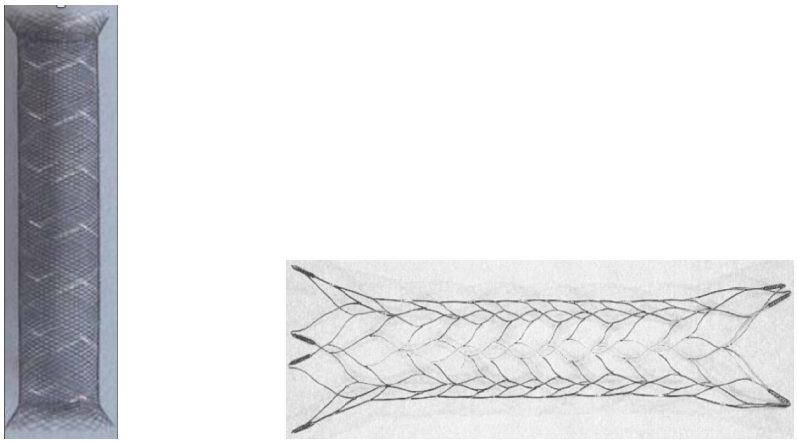


Fig.3.8 Example of a flow diverter & stent

Concretely a FD can be described as a porous tubular structure that is delivered through a catheter at the location of an aneurysm and its role is to divert the blood flow away from the aneurysm and thus diminish the pressure on its wall, preventing the aneurysm to grow further and, in the best case to recover its normal shape.

In paper **[7]**, we can also find a justification for the idea of working with a flow diverter like a stent without coils and an introduction of the reason for studying its utility… "The goal of this treatment is to disrupt the inflow to the aneurysm and cause a complete thrombosis of the aneurysm. This requires new stents specifically designed to achieve this goal."

…"These studies have shown that stents can produce significant alterations in the intra-aneurysmal blood flow patterns, but it is still not clear whether these alterations will cause complete occlusion of the aneurysm and thus be a viable durable treatment."

The methodology for image-base simulation of blood flows in stented cerebral aneurysms can be divided into the following stages **[7]:**

- Vascular modelling
- Stent deployment
- Meshing
- Hemodynamic modelling
- Data reduction and visualization

The two parts that concern my thesis are the stent deployment and the meshing. If we look into more detail these two parts, the stent deployment consists in the following steps:

1. Extraction of the vessel centreline
2. Deformation of a cylindrical support surface along the vessel skeleton until it conforms to the vessel walls. Mapping of stent designs using the deformed support surface

The meshing: the traditional computational mesh, which is generated in order to simulate the flow of blood inside the parent vessel and in the presence of a stent is called a body-conforming mesh.

Also in this paper we can obtain the justification of one of the assumptions that we made in the project, vessel walls are assumed rigid (page 3571).

"…Sensitivity analyses conducted using vessel deformations measured with dynamic imaging techniques show that flow characteristics such us size and location of the flow impaction zone and intra-aneurysmal flow patterns do not change significantly in comparison to simulations conducted using rigid walls."

During the work presented here:

- The vascular model was segmented from 3D CT data using Slicer 3D
- The flow diverter positioning was achieved with a self made program written with MATLAB
- The meshing of the flow diverter was also realised with MATLAB and then imported in ICEM CFD, which was then used for obtaining the mesh of the hemodynamic volume comprised between the vessel wall and the flow diverter.

4.2 Technological background

4.2.1 Virtual endoscopy

A virtual endoscopy is a “method of diagnosis using computer processing of 3-D image datasets to provide simulated visualizations of patient specific organs similar or equivalent to those produced by standard endoscopic procedure. Besides the standard endoscopy is invasive and uncomfortable for patients. They sometimes have serious side effects such as perforation, infection and haemorrhage” **[6].**

See in Figure 3.9 an example of a virtual endoscopy. First there are the different slices that compose the CT (from a colonoscopy). Following, there is a cube with the volume of a colon, then the segmentation of one part of the colon and finally the extraction of the centreline of the part of the colon segmented

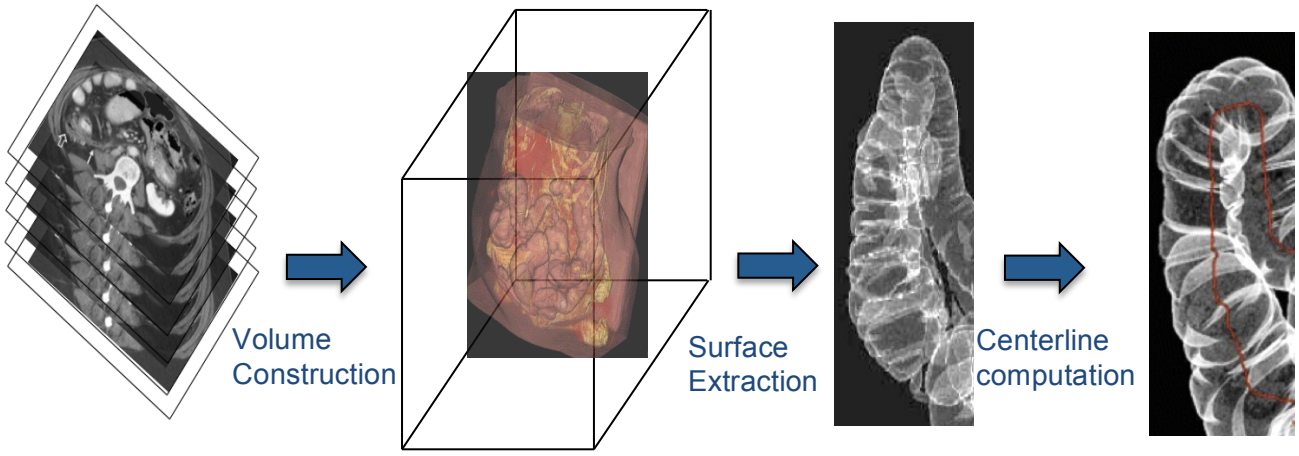


Fig. 3.9 Virtual endoscopy process, example of a colon

4.2.2 Advantages

It allows new types of viewing control and possibilities that are impossible by using real endoscopy. For example different scales of view, more clearly images, new measurements

are possible, etc. There are many body regions not accessible to real endoscopy that can be explored with virtual endoscopy **[6].**

Besides the improving of these methods, it is now possible to detect an illness more easily: Growing evidence shows that early detection of cancer can substantially reduce mortality, necessitating screening programs that encourage patient compliance. The most promising role for virtual endoscopy is in screening for colorectal cancer **[6].**

Two methods exist for post processing of data: surface rendering and volume rendering.

In the next table, table 3.1, there is an example of some advantages and disadvantages of a virtual colonoscopy:

Advantages	Disadvantages
Non invasive	Difficulty visualizing areas of viscous secretions
No sedation required	Difference depending on source image quality
Fast sensitivity	Does not allow biopsy specimens to be taken
It enables an evaluation of the inner inner surfaces of anatomic structures	Accurate delineation can be extremely difficult

Table 3.1. Advantages and disadvantages associated with virtual endoscopy **[1]**

4.2.3 Calculation of the centreline

There are different geometrical methods to find the centreline or curve skeleton of a vessel, defining the curve-skeleton as a method to capture the essential topology of the underlying object in an easy to understand and very compact form **[4].**

Another important definition is the medial axis/surface of a shape that it is a set of curves defined as the locus of points that have at least two closest points on the boundary of the

shape. The major disadvantage of the medial surface is its intrinsic sensitivity to small changes in the object's boundary due to the way it is defined (page 2) **[4].**

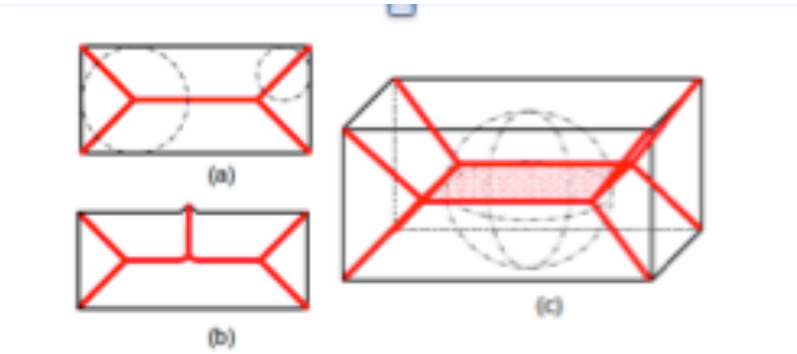


Fig.3.10 A medial axis in 2D (a and b) and a medial surface in 3D (c) and a few examples of inscribed discs (2D) and a ball (3D)

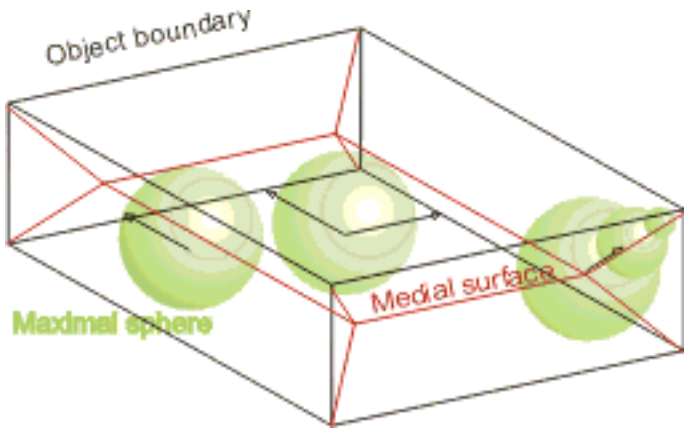


Fig.3.11 A medial surface in 3D (from http://sog1.me.qub.ac.uk/Research/medial/medial.php)

There are different methods to obtain the centreline, but the most important are **[4]**:

- Thinning and boundary propagation
- Distance field
- Geometric methods:
    - VORONOI Diagram
    - Cores and M-reps
    - Shock scaffold
    - REEB graph

Thinning and boundary propagation, this method consists to produce a curve-skeleton by iteratively removing voxels from the boundary of an object until the required thinness is obtained. **[4]**

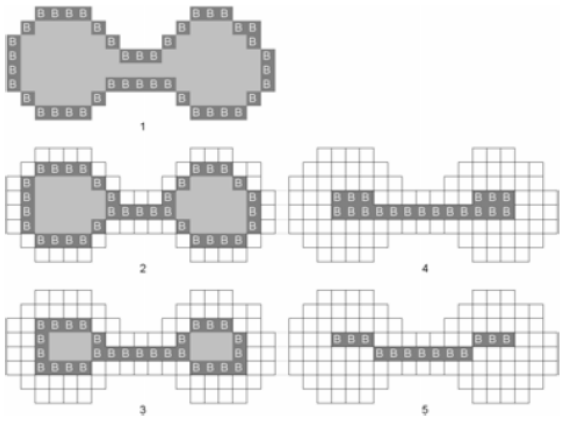


Fig.3.12 Example of the thinning algorithm

Distance field, the distance transform or distance field is defined for each interior point P of a 3D object O as the smallest distance from that point to the boundary B (O) of the object. **[4]**

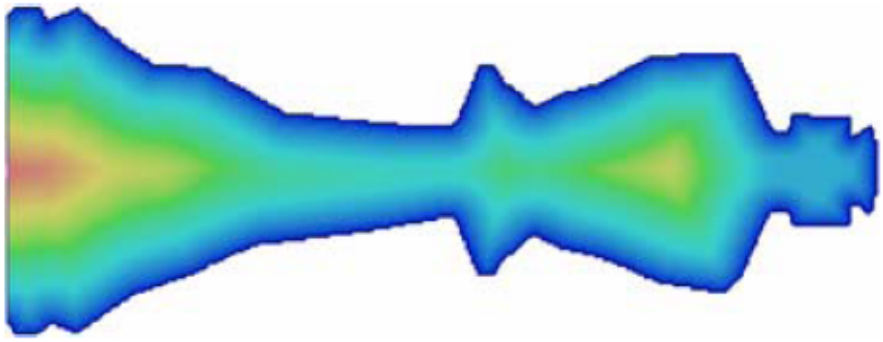


Fig.3.13 Example using the distance field algorithm

Geometric methods (page 9) apply to objects represented by polygonal meshes or scattered point sets. This is the kind of algorithm that I used because once I had my region with the aneurysm segmented I work with the surface meshed (triangular mesh). There are different types of geometric methods such as:

o VORONOI diagram that represents a subdivision of the space into regions that are closer to a generator element than to any other such element. The internal edges and faces of the VORONOI diagram can be used to extract an approximation of the medial surface. **[4]**

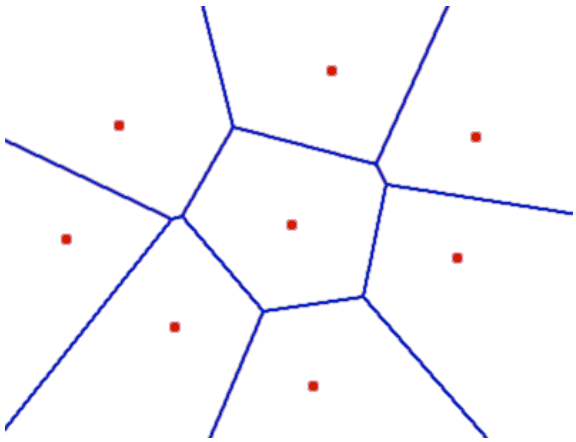


Fig.3.14 Example of a VORNOI Diagram

o Cores and M-reps are also medial axis/surface approaches. The location of the core represents the middle of the figure and the spread of the core represents the width of the figure. M-reps are a generalization of the Core concept using a “net” of connected atoms. Each atom has a position and additional information describing the shape locally. **[4]**

o Shock scaffold It is based on the notion of contact with spheres, for example the loci of spheres osculating sources and represents the medial axis/surface by a set of shock curves, defined as the intersection of medial surface sheets. “The resulting shock scaffold is a graph structure consisting of nodes and curve segments. Together with hyperlinks (surface patches), the shock scaffold gives rise to the shock hypergraph which is a complete representation of shape.” **[3]**

o Reeb graph based shape descriptions, with roots in Morse theory, are 1D structures encoding the topology and geometry of the original shape. The Reeb graph is not a curve-skeleton. The Reeb graph captures the topology of a compact manifold by following the evolution of the level sets of a real-valued function defined on the respective manifold. **[4]**

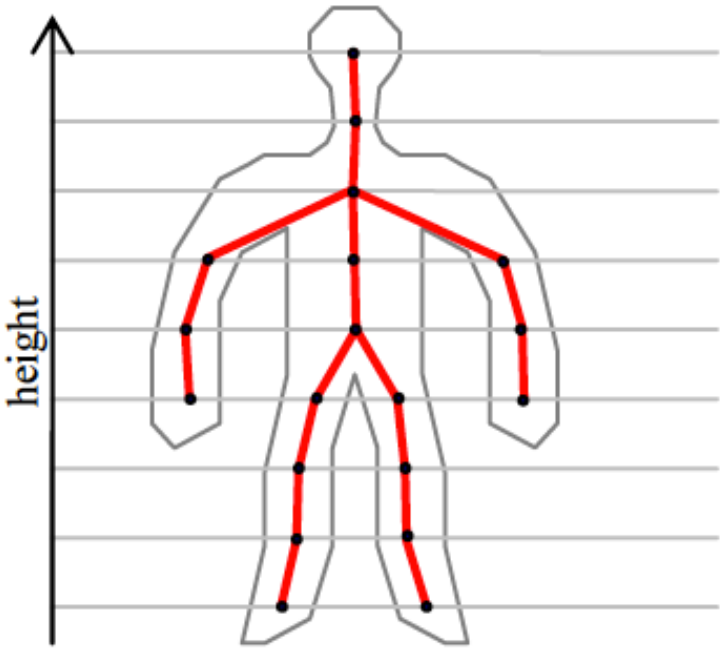


Fig.3.15 Reeb graph example

The main disadvantage of medial-axis based in geometric methods is their sensitivity to noise.

5. Contribution

5.1 Methodology

5.1.1 Pipeline

The pipeline that we have followed to obtain a surface representation of the vessel from the CT is similar to that of Figure 3.9. Specifically the steps were: to find all the slices that contain the aneurysm; then, to select the voxels that configures the aneurysm and, finally, to reconstruct the image to have a 3D shape of the aneurysm.

5.1.2 Tools

The tools used by requirement of the group in Zurich for the development of the project were:

- SLICER 3.4→ to segment the data from a CT
- PARAVIEW→ to cut different parts of the segmented data
- ICEM CFD→ to introduce the inflow/outflow surfaces and mesh de segmented data
- MATLAB→ to program the code to produce the virtual flow diverter

5.2 Procedure followed

Summary of the different steps followed to obtain the virtual flow diverter placed inside a vessel:

6.2.1 Three-dimensional Vessel Representation

6.2.2 Generation of Flow diverter

6.2.3 Output

The three first steps are represented in the next diagram:

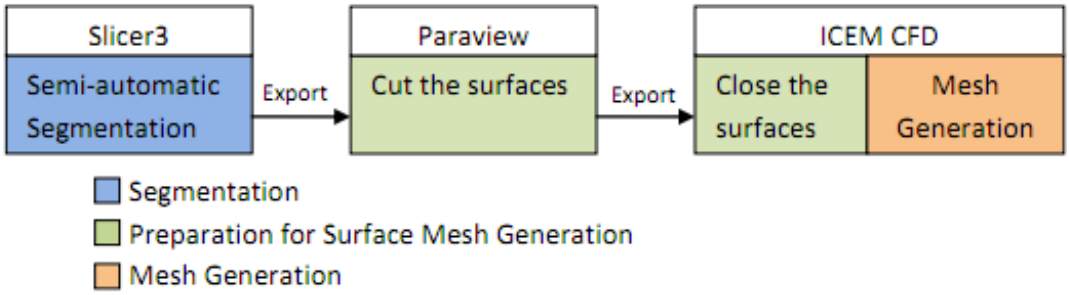


Fig. 4.1 Summary of the first three steps

5.2.1 Three-dimensional Vessel Representation

5.2.1.1 Segmentation

The main goal of this first step is to take a sample from a CT with contrast agent as an input and to make a segmentation of the piece of vessel with the aneurysm.

A CT is a non-invasive medical imaging method that produces a 3D reconstruction of the inside of a patient by processing a “stack” of 2D images. It gives characteristics of the internal structure of the patient such as dimensions, shape, internal defects and density.

This part of the process is done by using the program Slicer3.4: which is an open source biomedical visualization software (see section 2.Software packages and libraries). Chronologically the user has to:

- First of all load the data.
- To find the region that has the aneurysm, to define the ROI. (Go slice for slice until we find it).
- Use a semiautomatic method to segment the ROI, using the threshold tool to discard a part of the CT (exclude unneeded vessels) and manually optimization, selecting and deleting the voxels that we want/do not want.

The table 4.1 is contained in the DICOM-data, which is a file format that can contain many different data and information.

Patient's Name	-
Patient's Sex	M
Patient's Age	61Y
Patient's Size	1.723
Patient's Weight	84.18
Slice Thickness	0.369
Reconstruction Diameter	512
Distance Source to Detector	1197
Patient Position	HFS
Frame of Reference VID	-
Samples per Pixel	1
Rows	512
Columns	512
Pixel Spacing	0.369
Bits Allocated	16
Bits Stored	16

Table 4.1 CT information

Brief overview of how the Slicer3 program works:

When the user starts the program, she/he'll see the main window (Figure 4.2) where there are four subwindows, a 3D view and three 2D view, the axial, sagittal and coronal orientation:

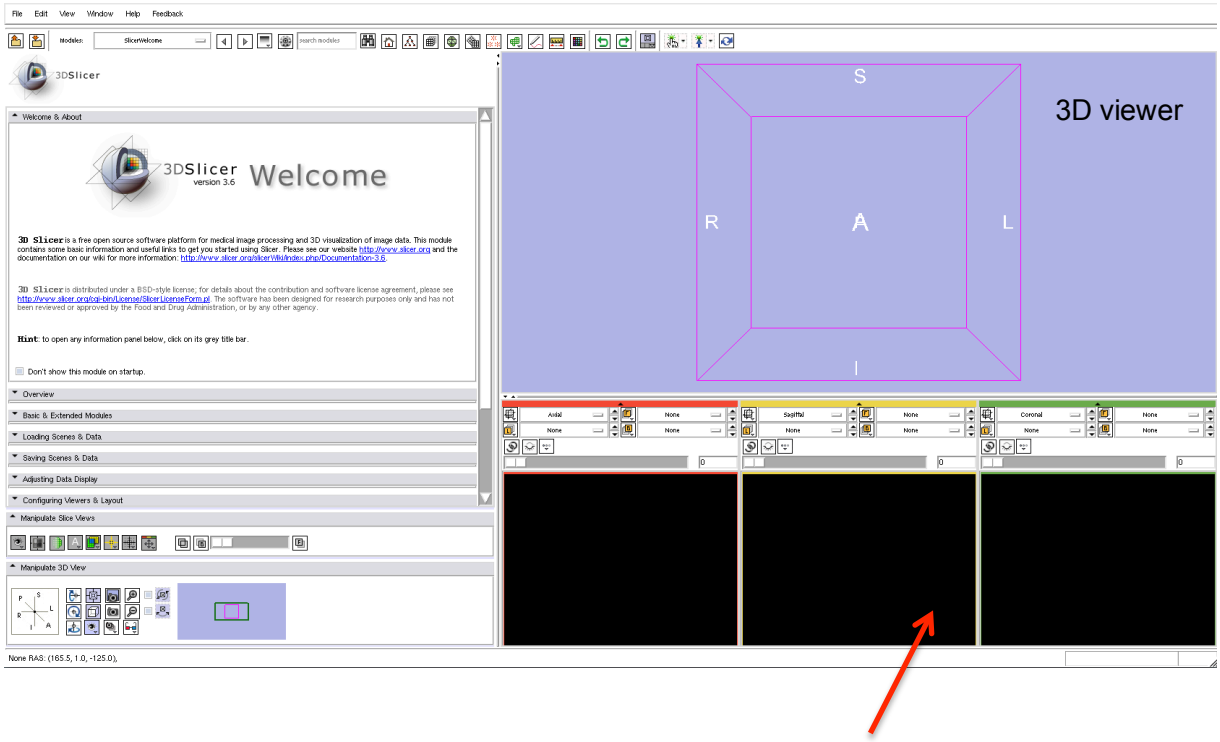


Fig.4.2 Initial screen when the users open the program

After loading the data from the CT (load the data as a volume, and choose the option centred, to have the image in the centre of the screen), the user can browse through the different 2D views to locate the aneurysm. On the example presented here, the aneurysm was located using the sagittal view of the brain (Figure 4.3).

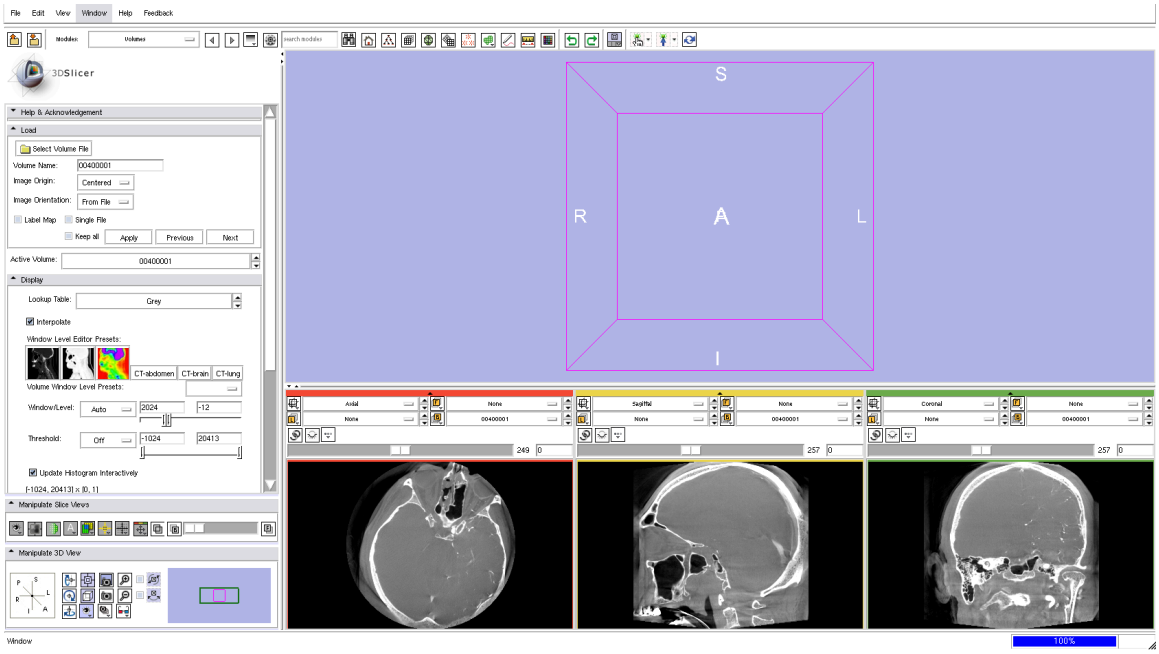


Fig. 4.3 The data loaded; see the images in the 2D viewer.

This manual process requires some time until the user founds the aneurysm, which it looks like a tiny balloon. See Figure 4.4 to have an idea of how it looks. The aneurysm is located in the middle of the image.

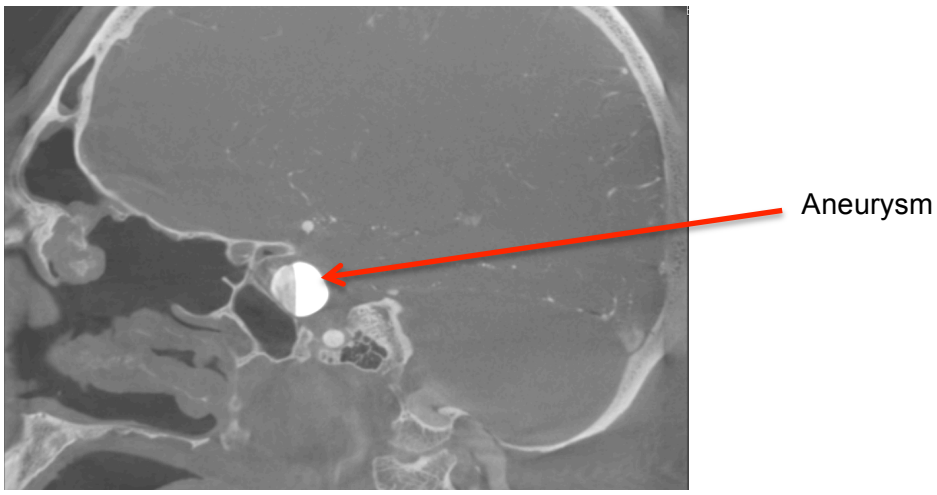


Fig. 4.4 Slice number 349 from the CT

Then, the user uses the threshold tool to restrict the visualisation to voxels having a defined density and thus sees more clearly where the aneurysm is. The threshold tool is used to select voxels that are in a specified range of intensity.

To use the semiautomatic method to segment the ROI, using the threshold tool, the user has to create first a “Label Map”. A label map is a group of pixels where every pixel has a value representing the anatomy present at that point. Table 4.2 shows the range of the threshold tool used in the procedure.

Threshold range	900.96	5888.3

Table 4.2 Threshold range

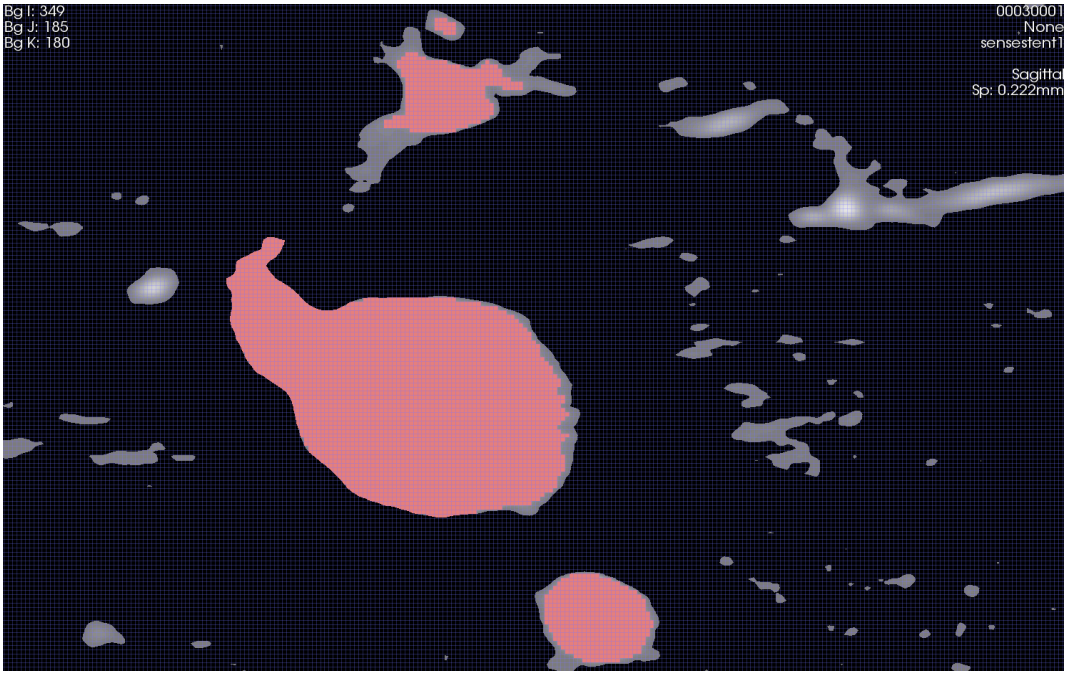


Fig.4.5 a slice of the CT showing the parts selected using the threshold tool

Figure 4.5 shows an example of one slice after using the threshold tool. The red parts are the selected parts that belong in the specified range.

As shown in Figure 4.6 there are selected parts of the vessel that we want and also taking some parts that are sticking out of the region of interest. For this reason the user (after using the semiautomatic method) has to manually select the voxels that wants and delete the others.

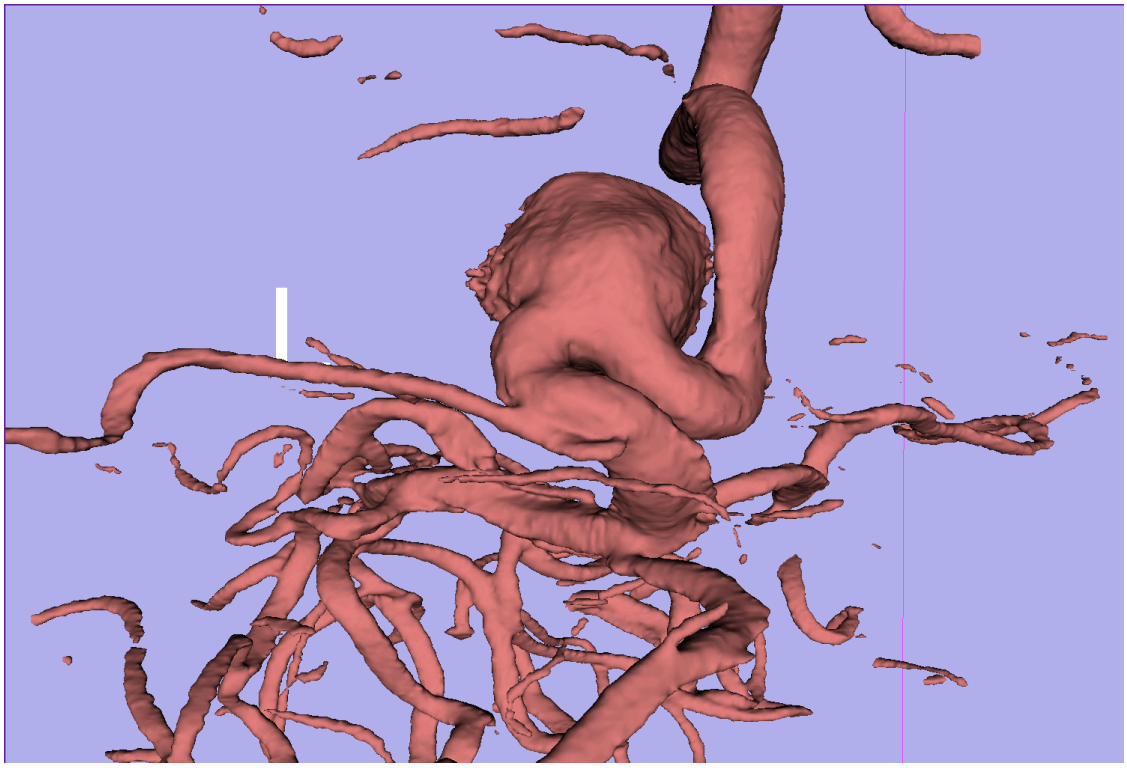


Fig.4.6 A 3D view during the segmentation process

Using the procedure previously described, there is no possibility to directly select the piece of vessel that we want. This is because by imposing a range of values for the threshold tool, we select different vessels in the CT that are in this range. That is why we need a manual optimisation.

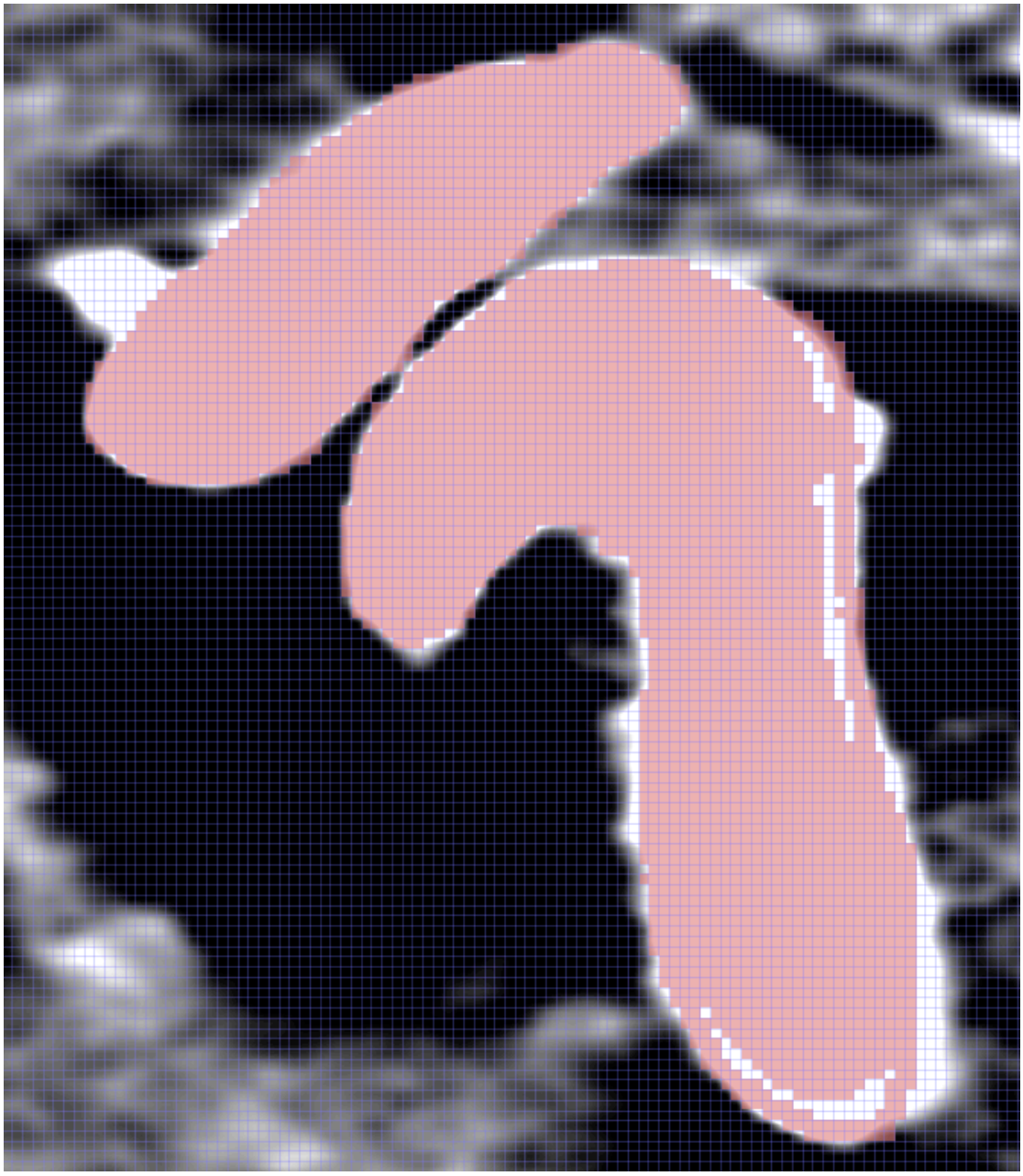


Fig.4.7 a zoom to see better the voxels selected.

In Figure 4.7 the pink voxels are the ones that belong to the vessel that we want to study.

Now the user has to select manually the white voxels that were not selected and belong to the vessel, using the “paint” tool or “draw” tool.

After all the procedure we obtain the part of the vessel with the aneurysm used as an input for the next steps:

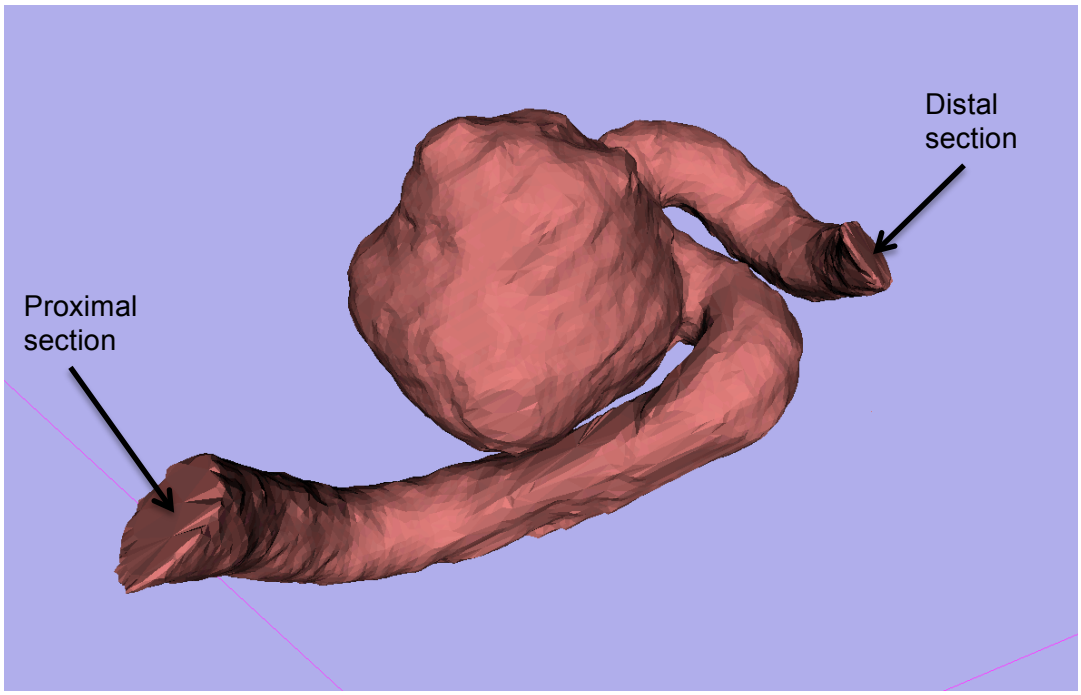


Fig. 4.8 Output after the segmentation using Slicer3

Before meshing the volume notice that the proximal and distal section of the vessel are not smooth enough causing distorted elements of the surface. For this reason, we need to cut this parts and include the new distal/proximal surface. The cuts have to be perpendicular to the centreline to later use the covers as an input/output for the different flow simulations and the utility of a closed surface it'll be explained later in 4.2.2.2.2 Partial Vessel Segments.

5.2.1.2 Handling of the sections

In this part, the program PARAVIEW (see section 2.Software packages and libraries) is used to cut the proximal/distal sections and then use another program called ICEM CFD to include the new proximal/distal sections surfaces and mesh the new surface.

The different actions that the user has to follow when using the PARAVIEW program are:

- Importing the segmented data from Slicer
- Cutting the proximal and distal sections of the data

Pipeline Browser → add → Filters → Alphabetical → Clip (clip type) → apply

Figure 4.9 is an example using the tool Clip (type: box) to cut one part of the vessel:

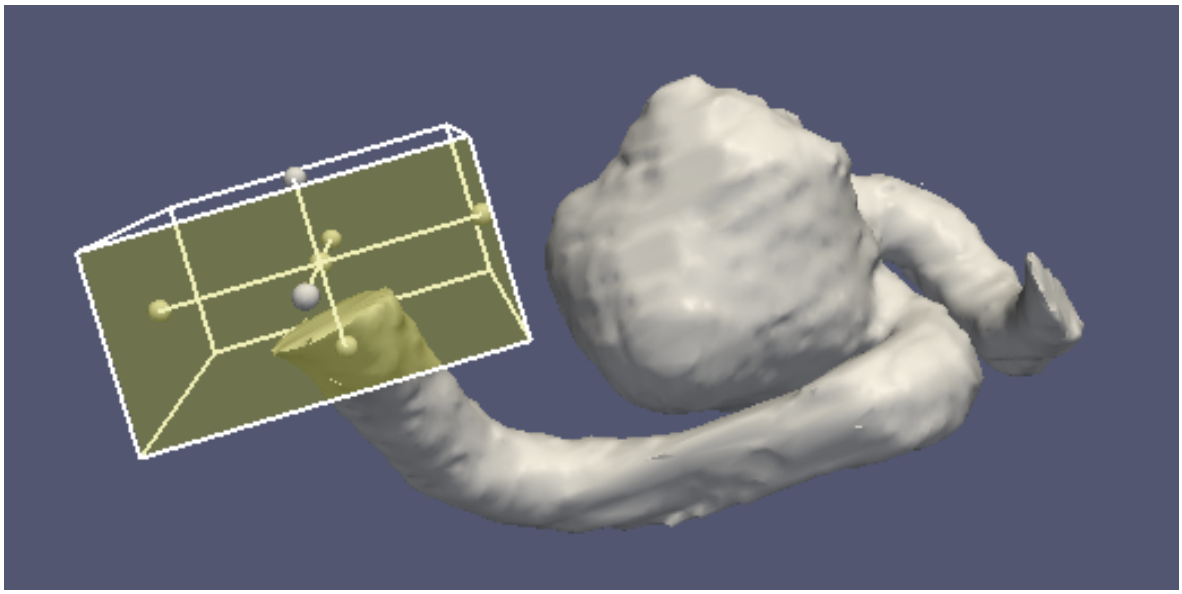


Fig.4.9 Box used to cut the proximal section

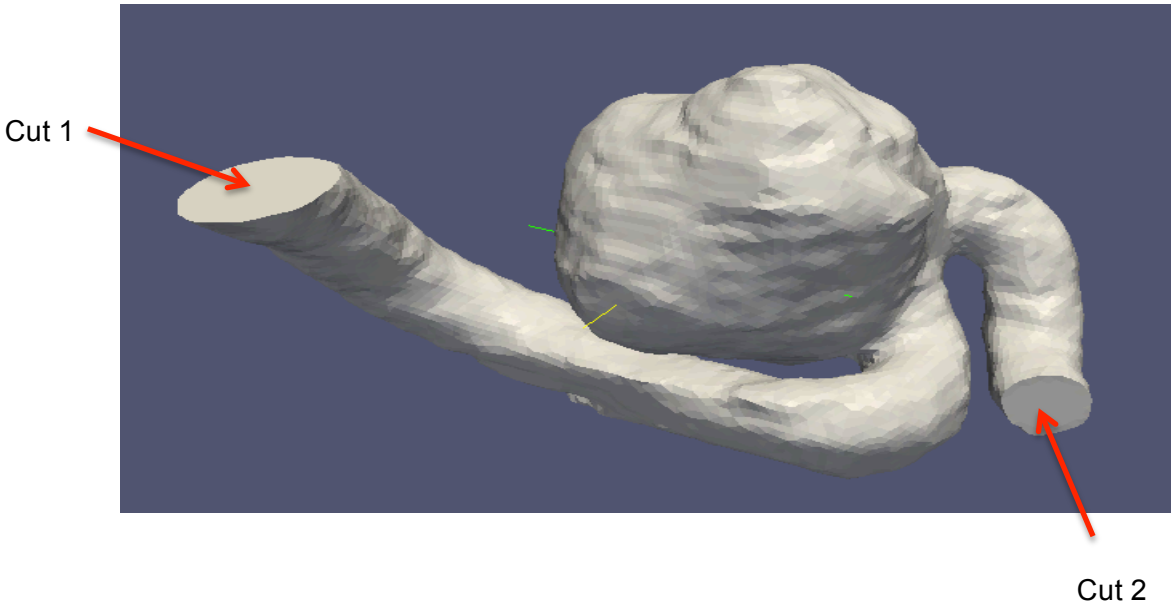


Fig. 4.10 Final result after cutting the surfaces

- Exporting the results to work with ICEM CFD. We had to extract the surface and triangulate it first in order to be able to export an STL-file.

Once the user has cut the vessel, he can use the program ICEM CFD to include the inflow and outflow surfaces and then mesh it.

Explanation of the method to include the covers (proximal and distal sections surfaces):

Load the surface → Under Geometry → Extract curve from surface (number of segments = 0) → Create/Modify surface → 1st option → Enter (Method: from curves)

5.2.1.3 Mesh generation

Explanation of the method used to mesh the surface:

Mesh → Global Mesh Setup → modify in Global Mesh Parameters: Global Element Scale Factor: Scale factor and Global Element Seed Size: Max elements → Compute …

The next figures, Figure 4.11 and Figure 4.12 show in more detail the command windows of the program and the fields that need to be modified.

The values that you will see there can vary depending on shape of the data segmented. For example the user may be more interested in a bigger elements of the mesh or with different kind of elements as quadratics, etc.

The main point is to choose the elements small enough to catch the geometry with sufficient details but not too small to keep the number of elements low.

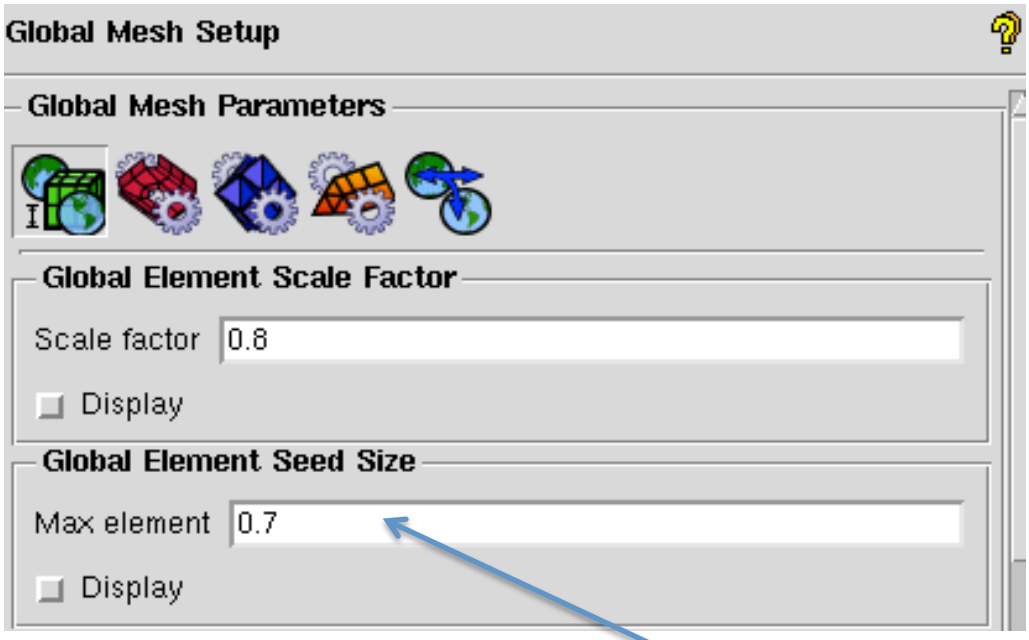


Fig. 4.11 Variables to change I

…→ Surface mesh → Select: Mesh type: All Tri and Patch Independent

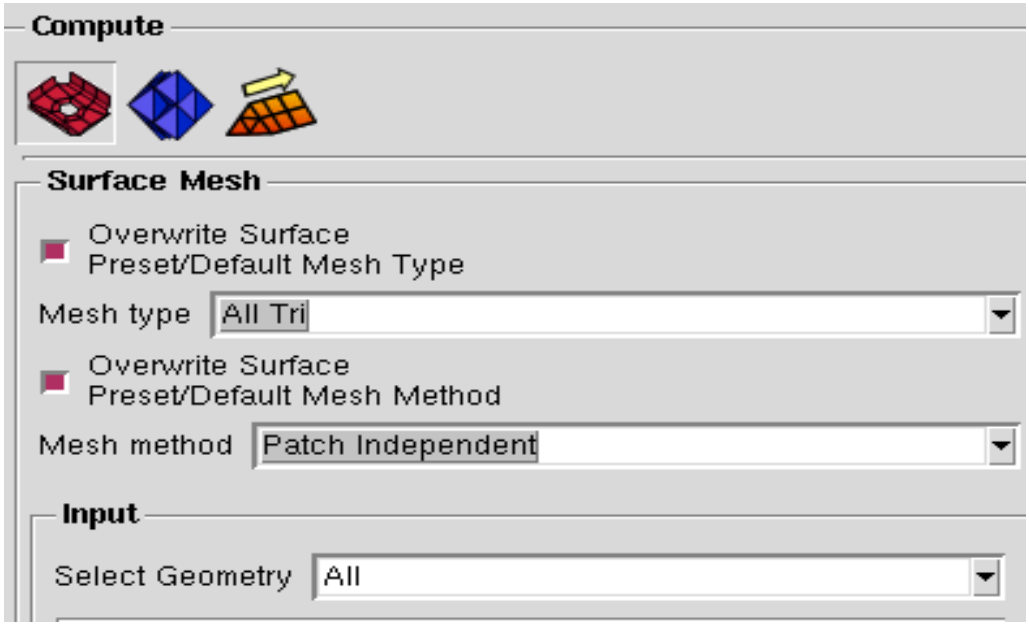


Fig.4.12 Variables to change II

At the end of this process, we obtain the surface mesh.

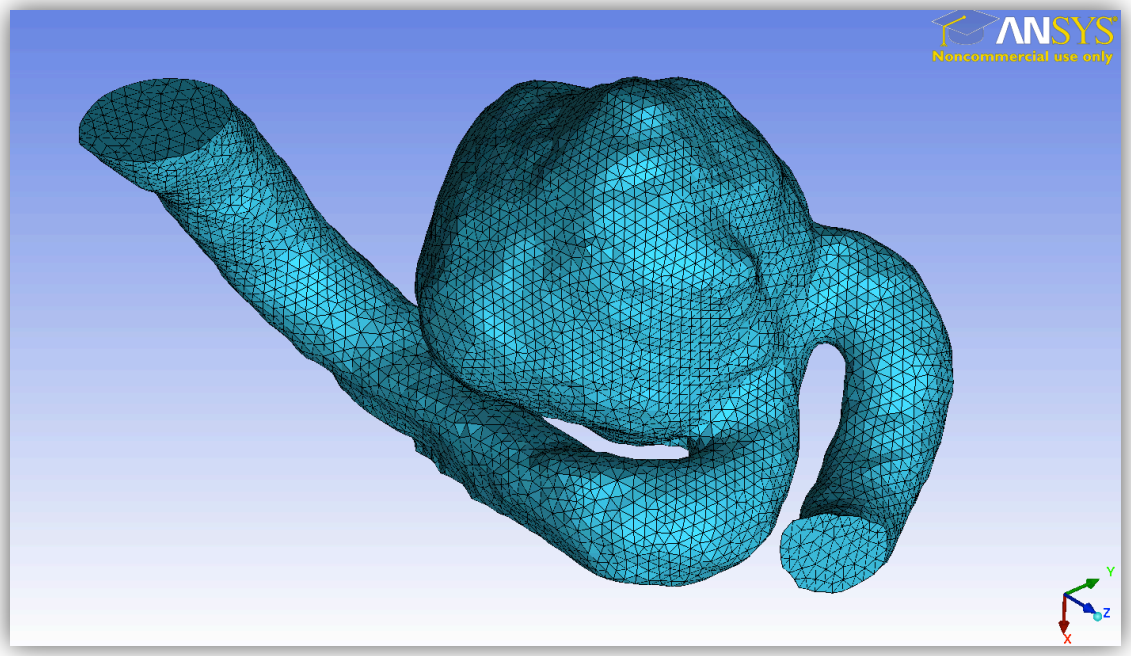


Fig.4.13 The surface mesh used for the next steps

Characteristics of the element mesh:

Faces	14058x3
Vertices	42174x3
Normal of the faces	14058x3
Ø of the vessel	5 mm
Length of the sack of the aneurysm	18 mm

Table 4.3 Surface characteristics

- Faces: It is the element matrix. Every row has the index of the vertices that form one triangle.

- The matrix Vertices contains the coordinates of the nodes of the triangles that conform the mesh. Every three rows it gives us all the coordinates of the vertices of a triangle.

- Normal of the faces gives the different normal vectors of the plane formed by the triangles of the mesh.

5.2.2 Generation of Flow Diverter

In this section the strategy followed to develop the virtual flow diverter will be explained from an easy case to a more complicated one in a way to make it clear.

In case of a straight flow diverter, see Figure 4.15, the flow diverter is the combination of different coils. If we just study one coil, because the other coils are do it with the same method, in every segment the coil along the centerline is done using a unique coordinate axis system, see Figure 4.14.

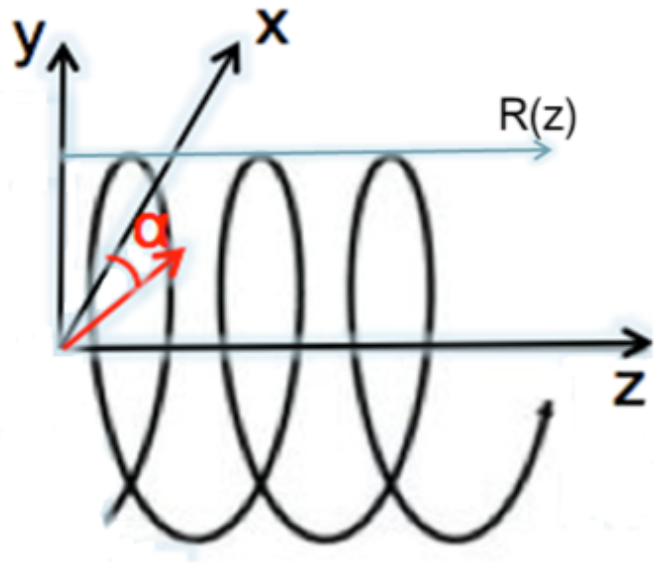


Fig. 4.14 One helix

$$x = R(z) * \cos(\propto (R(z)) \quad \textbf{(Eq.4.1)}$$

$$y = R(z) * \sin(\propto (R(z)) \quad \textbf{(Eq.4.2)}$$

$$z = b * t \quad \textbf{(Eq.4.3)}$$

α = angle between coils, α depends on the number of coils. The coils are defined with points placed at a distance R(t) on the plane E^*_1=0.

Also say that the pitch (distance between two picks) is:

$$Pitch = 2\Pi * b \quad \textbf{(Eq.4.4)}$$

And having an example with more coils, the result is:

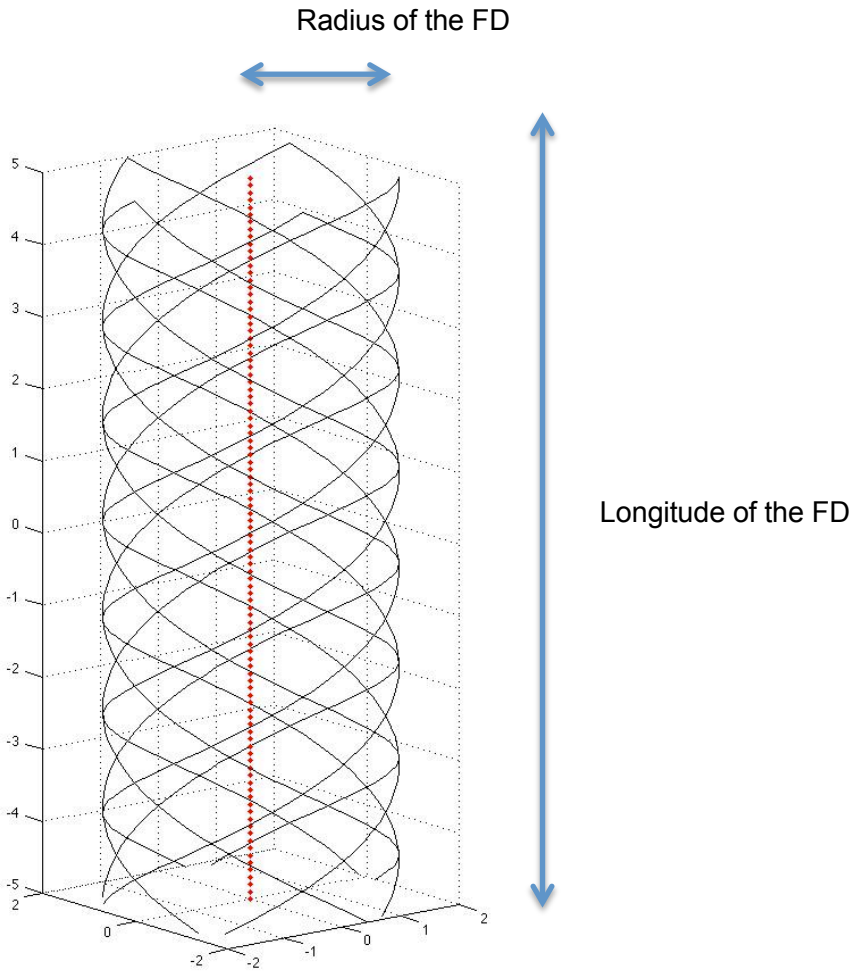


Fig. 4.15 Straight FD

In Figure 4.15 a virtual flow diverter is reproduce using the MATLAB script called Produce1DStraightStent.m, see ANNEX B. The different parameters used to reproduce the FD are:

- Radius of the stent = 2
- Longitude of the stent =10
- Number of turns = 10
- Number of points on the midline = 100
- Number of coils = 10

In a more complicated case where the centreline is not straight anymore, a unique coordinate axis system will not work. We need to define a coordinate axis system for each point of the centreline and be sure that the radius of the FD along the centreline is adapted to the vessel in a way it is not sticking out. Besides a function along the centreline to control that the radius fulfils the different criteria is necessary.

The software used in this part of the thesis is MATLAB (see section 2.Software packages and libraries). It is used for the extraction of the centerline and the creation of the FD.

Basically the program is structured in four parts:
- Initialisation
- Centreline construction
    - Flagging
    - Partial Vessel Segments
    - Create the centreline
- Flow Diverter Envelope
    - Nominal Coordinate Axis System (CAS)
    - Creation of the flow diverter envelope
- Optimisation

Initialisation

Set the global values from our input “STL” file:

[F,V,N] = STLREAD(FILENAME) returns the faces of the triangles (F), the vertices (V) and the face normal vectors (N) in different matrixes.

A short introduction of what is an element will be explained below: it is a triangle, created with three points (V) that has one normal, the normal of the plane that contains the triangle (N) and one face (F):

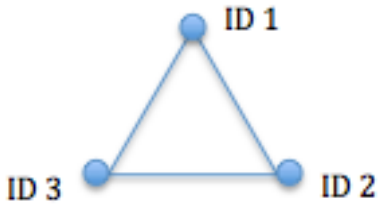


Fig.4.16 One triangle-element

In our case F refers to the element matrix, for example the first line of the matrix gives the indices of the vertices of the first element. V is the vertices matrix that gives the position of each vertex referenced in the global coordinate system.

STL format: *STL files describe only the surface geometry of a three dimensional object without any representation of colour, texture or other common CAD model attributes. The STL format specifies both ASCII and binary representations.*

In the example that we are following in all the thesis:

F	14058x3
N	14058x3
V	42174x3

Table 4.4 STL data

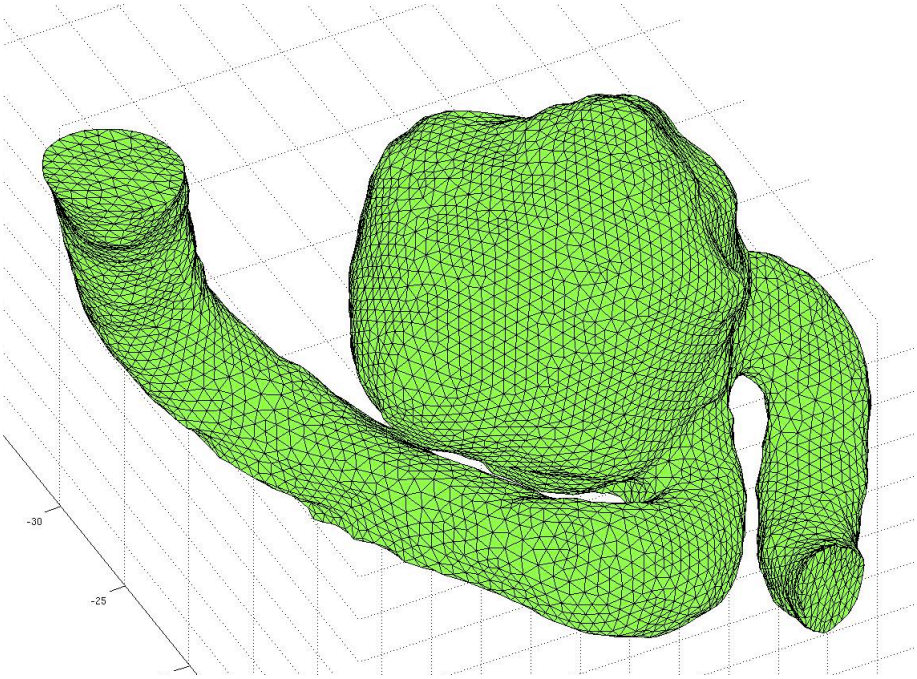


Fig. 4.17 Vessel after plotting the STL file with MATLAB

Centreline construction

- Flagging

Flagging is a procedure that is used to differentiate the different elements of a surface mesh depending on their position using an index like 0,1,2.... In our case we differentiate between the elements of the proximal section surface, the distal section surface and the vessel wall. The utility of this is for later when we find the centre of mass (CM) because we use two different methods to find it depending in which part we are and using the flags we avoid to take unnecessary elements of the mesh.

First of all, the user has to select the proximal section surface (first cover) and the distal section surface (end cover) to “flag” the different triangular elements, Flag=1 for the first cover, Flag=2 for the end cover and Flag=0 for all other elements. Using this method, now the program can differentiate the different parts of the vessel and take just the elements that It wants depending the investigated region. Example of this first part, Figure 4.18, selection of the front cover, the user just has to select the contour that he wants with the mouse (interactive part between user and program):

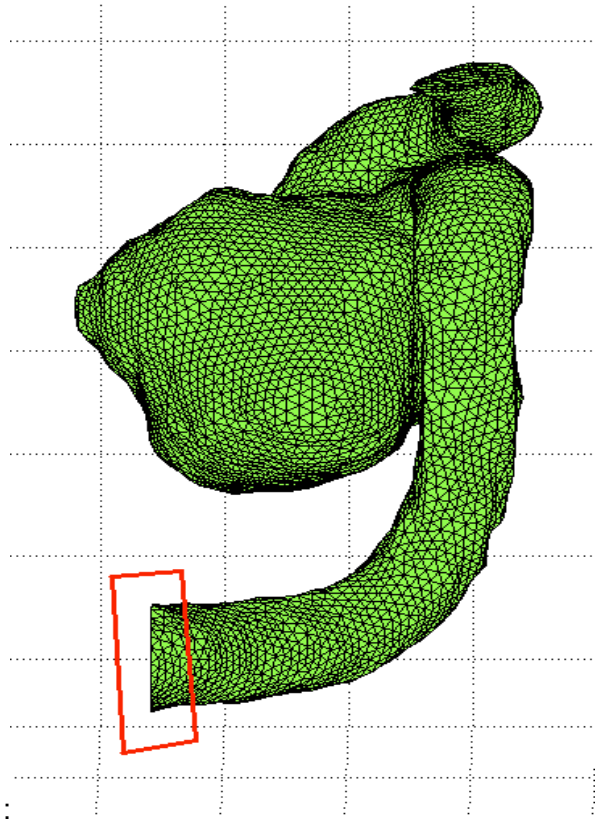


Fig.4.18 Selecting the first cover

After selecting the region near the section surface (see Figure 4.18), the colour of the region that was selected changes, just to show which are the elements that the program takes (see Figure 4.19). Then the user has to select one element of the section surface (pink element in Figure 4.20) to take all the elements that have the normal vector parallel to that element, this condition ensures that it takes only all the elements of the section surface.

Just for check if the elements of the covers are well selected the program plots the elements again (see next page):

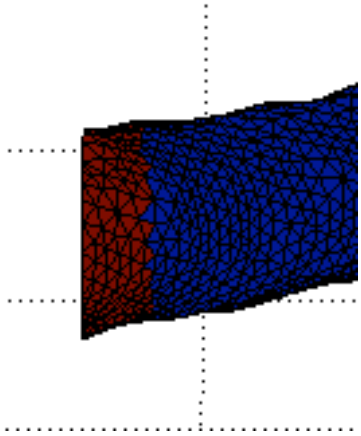


Fig.4.19 Region selected

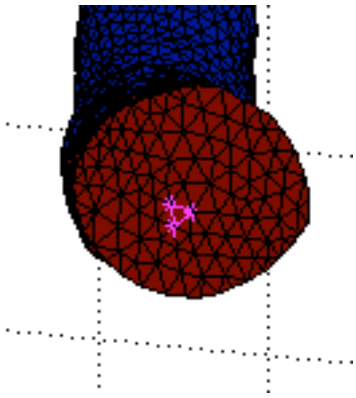


Fig. 4.20 Selecting one element

Once the program executes this part it asks to the user what to do next:

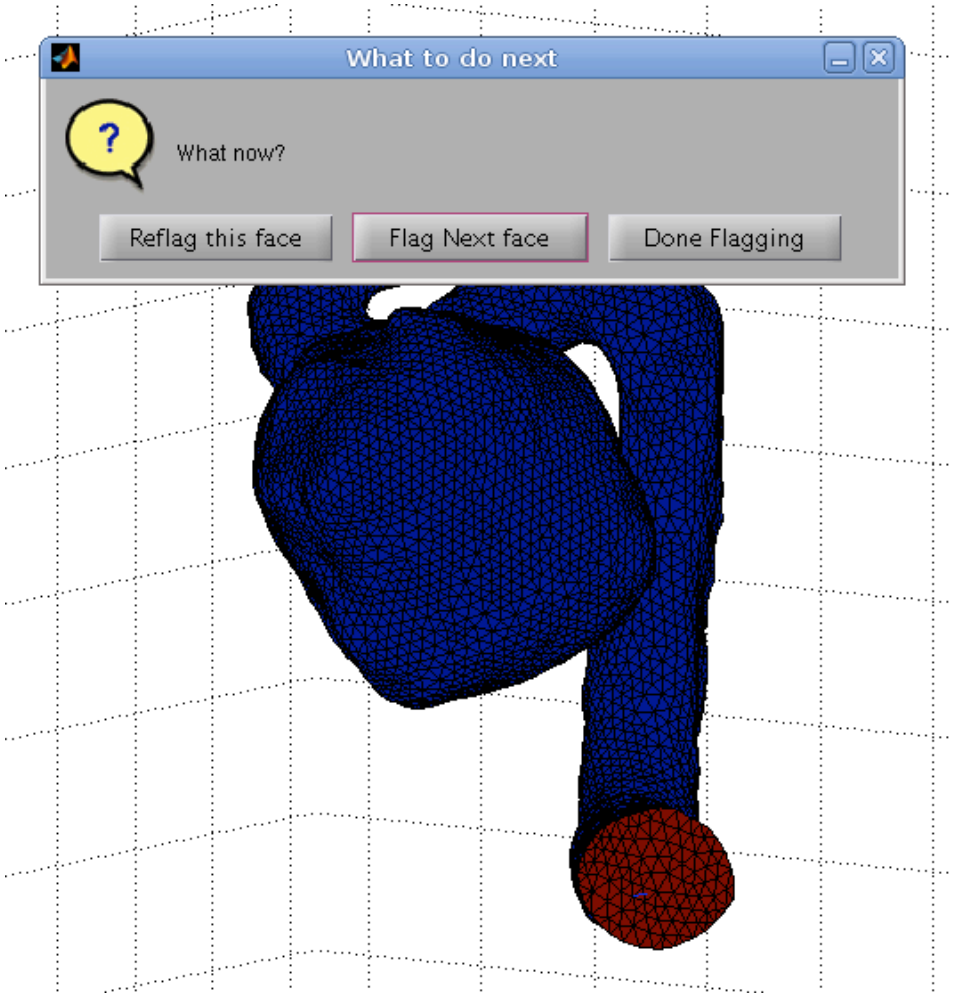


Fig. 4.21 Box dialogue (1)

The user can choose between “Reflag this face” (in case the user selects something wrong),“Flag next face” to continue flagging or “Done Flagging” if the user has flagged all the faces.

In the next example the user select: “Flag next face” so he has to select the end cover:

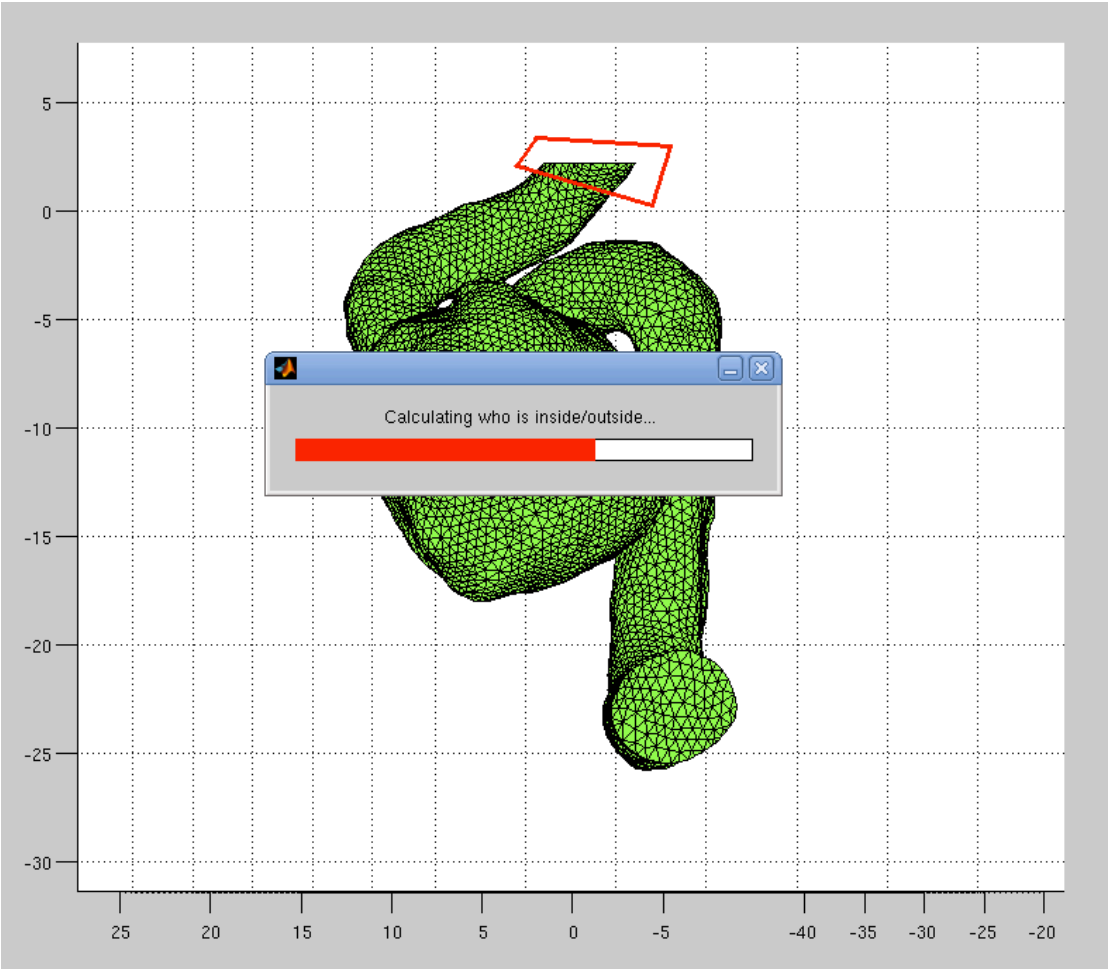


Fig. 4.22 Example selecting the second cover

A waiting bar always appears to have an idea of the time left until completion of the process.

At the end of this part, the program save a variable called "GlobalFaceFlag”, that contains a matrix with only 0’s, 1’s and 2’s depending if the elements are in the covers or in the bounding of the vessel.

- Partial Vessel Segments

It is in this part where the program finds the centres of mass (CM) of the vessel. The utility of finding the CM is then to create the centreline as the union of all the CM's and have a reference inside the vessel to implement the virtual flow diverter.

To perform that, the user has to go through the vessel selecting different parts of it, using a bounding box (BB).

It is important for the user not to include the cover inside the BB because another method is used to find the CM in the covers (just take all the triangles that are in the covers and find the CM using the vertices of each elements).

The function implemented is the function "GetCoM". It uses as an input the "GlobalFaceFlag" that was found in the step before. The functionality of this function will be explained in more detail in page 59.

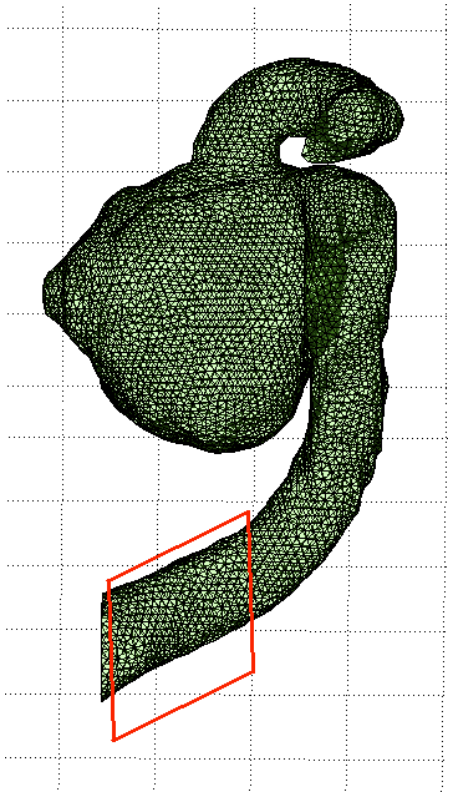


Fig. 4.23 Example of the BB

In Figure 4.23 the user is selecting one region of the vessel to find the CM, the red rectangle is the BB. The result after selecting one part of the vessel with the BB is the CM in that region:

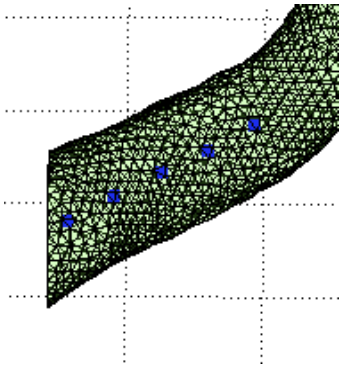


Fig. 4.24 CM in the region selected with the BB

Always after selecting one region the user has different possibilities to follow:

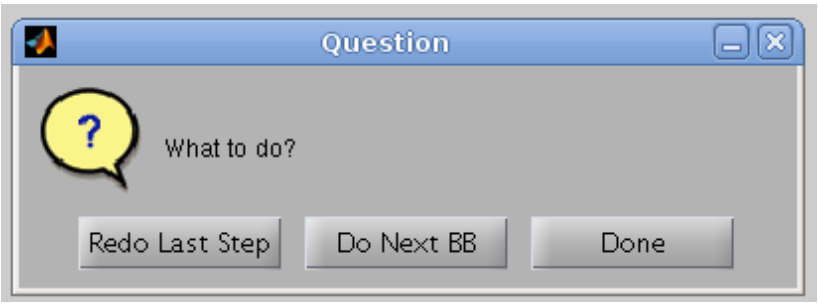


Fig.4.25 Box dialogue (2)

The user can choose between “redo the last step”, to find again the CM, “do the next BB” to find the next CM or “Done”.

If the user chooses “Do Next BB”, he can select a new region to find the CM. This procedure is repeated until the entire vessel was processed.

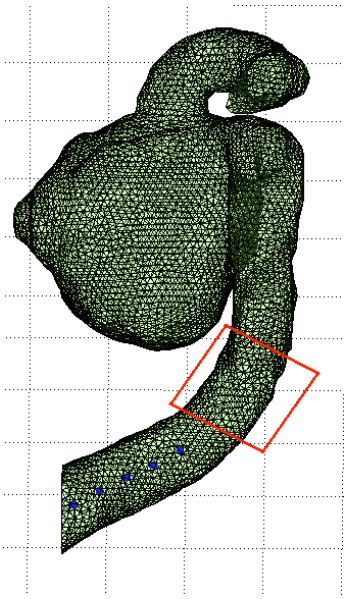


Fig. 4.26 Select a new region

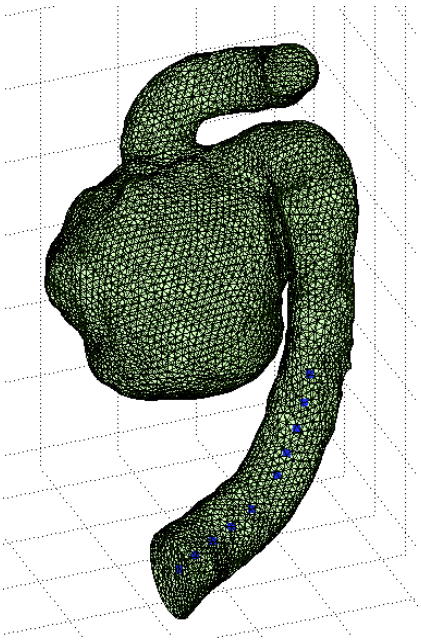


Fig.4.27 The new CM's

During this procedure the volume contained in the Bounding Box is populated with points organised in a grid-like manner. In second phase, the function "InPoliedron" detects the points inside the manifold closed surface selected with the BB. Finally, these points are used in the function "GetCOM" to find the CM, based on Equation 4.5. This is the reason why a closed surface has to be used.

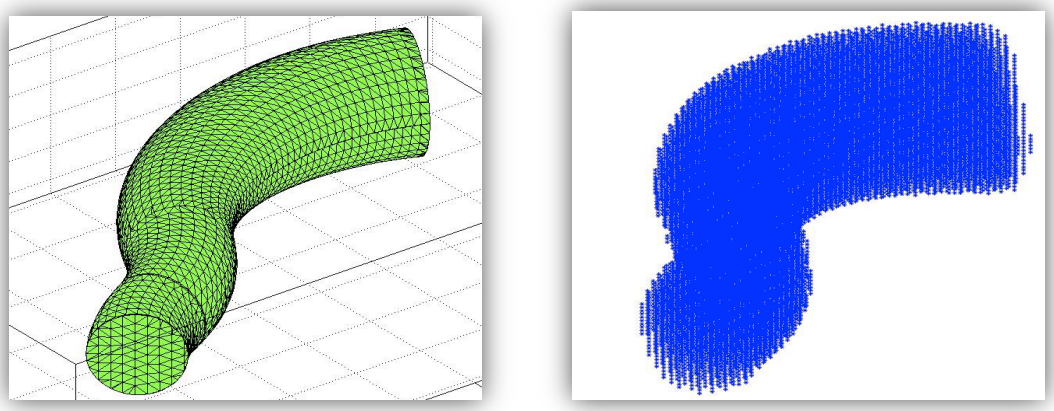


Fig.4.28 Example of the points that are inside the BB in a piece of a vessel.

After the all procedure we obtain the CM's of the vessel except in the neck of the aneurysm:

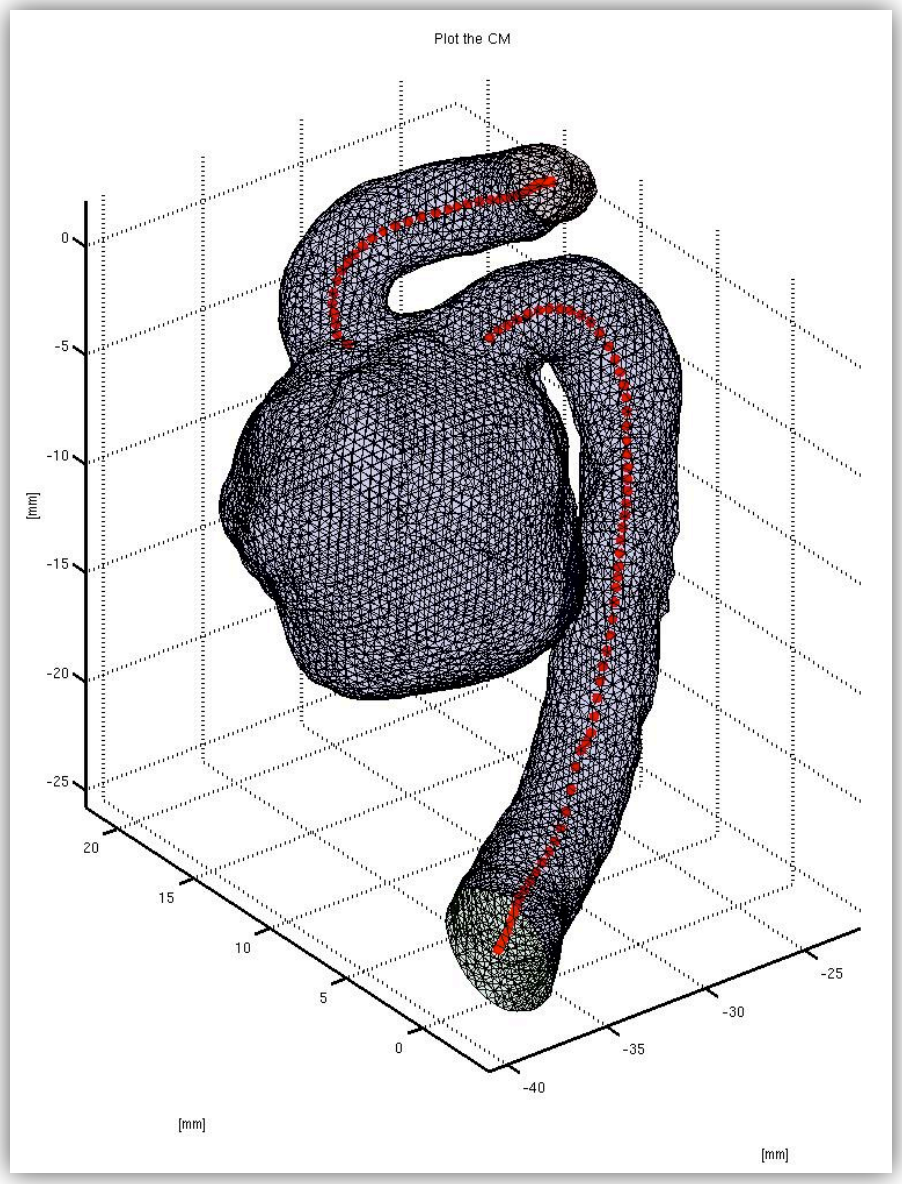


Fig. 4.29 Partial CM. Please notice that there are no points in the neck of the aneurysm

To understand better why we can not use this procedure to find the CM in the neck of the aneurysm, we need to enter in more detail with the function "GetCoM".

With this function we make a discretization part-by-part creating different bounding boxes and making a 3D rasterization for each BB to find the CM.

Formula to find the CM:

$$CM = \frac{\Sigma mi * pi}{\Sigma mi} \quad \textbf{(Eq. 4.5)}$$

Once the user create a BB a reorientation of the axes inside is made, having the X axes parallel with the edge of the rectangle and made the rasterization along the X axes:

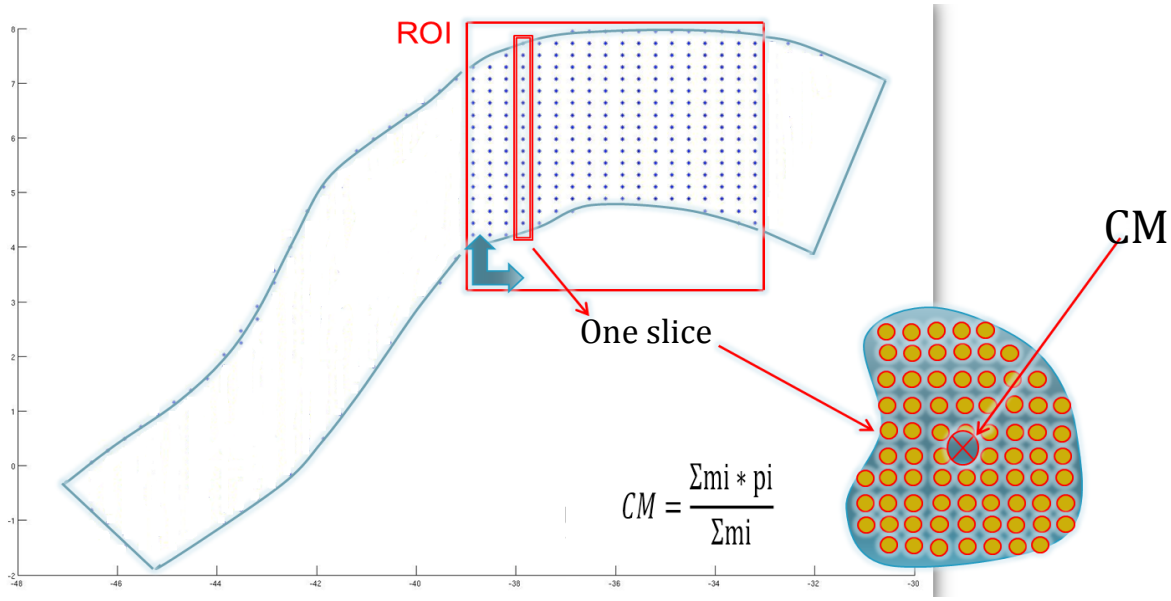


Fig. 4.30 Example during the rasterization process

For this reason we can not use this method to find the CM's in the neck of the aneurysm because if for example we take a slice in that part, we'll have all the points of the vessel and also the ones that belong to the aneurysm, causing a drastic change of the centreline orientation.

The numbers of points that are created in the BB are regulated with three different parameters controlling the X, Y and Z spacing of the points used inside the BB. By increasing or decreasing these parameters the accuracy and the number of CM along the centreline can be changed.

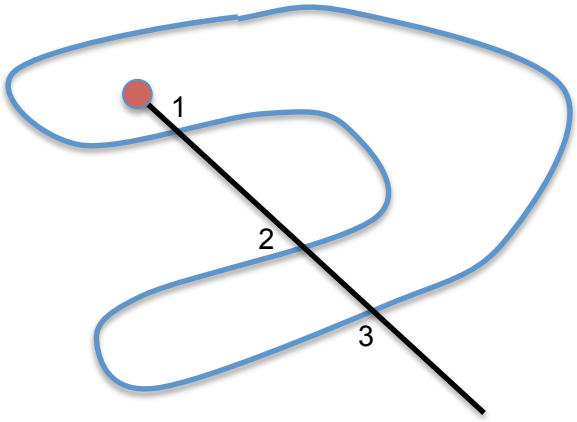
If the bounding box is big enough to produce 2 or more CM, the CM will be created equidistant but if just one CM can be created then this will be placed in the middle of the BB.

It is interesting to know how the BB is created. The idea is to make one point on one side of the vessel and then another in the same side and following the direction of the blood flow,

just to have a reference. With these two points now we have one edge of the rectangle draw. Then the user has to create another point in the other side of the vessel and the program automatically closes the rectangle. The concept of the rectangle is very important because we make a rasterization a long the X axes so we need the edges parallels.

In general, to check if a point is inside the limits:

- From the given points a few rays are shot in different directions
- If the ray crosses an even number of planes in its way, it's outside else it's inside.



→ 3 crossings, odd number, the point is inside.

Fig. 4.31 Example of checking a point

One ray would theoretically be sufficient to make the check. Practically, more rays are used to exclude any false positive/negative that could occur in special cases.

To check if a ray crosses a plane:

- Find the intersection ray-plane in the given direction
- Check if this intersection is within the plane

Notice: the planes are infinite in the depth direction.

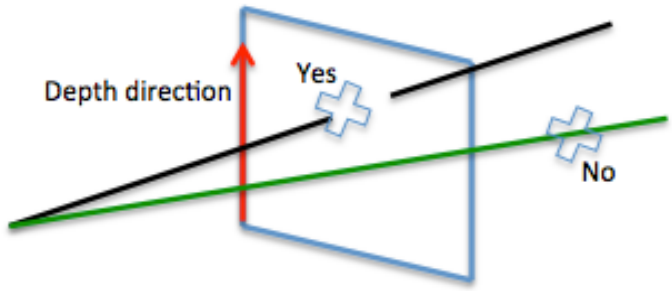
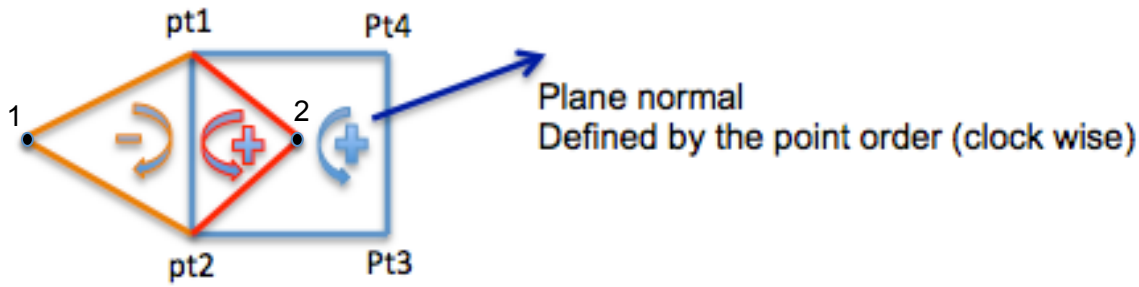


Fig. 4.32 Example of checking if it crosses

The user has to check if the direction of the plane is the same as the direction of the triangle formed by the intersection point and the plane corner.



Intersection 1 → outside

Intersection 2 → inside

Fig. 4.33 Example of checking the direction

- Create the centreline

A spline interpolation between all the CM's is done to create the centreline. However using this method we made an error in the neck of the aneurysm that we have resolved in the optimisation part.

In a first approach, we solved the problem by making a spline interpolation between the last point before the neck of the aneurysm and the first point after the neck of the aneurysm. As it's normal, with closer points better interpolation and a cubic spline line too long has more

error. Also a light smoothing of the curve is made to suppress small perturbations of the curve. Formula of the cubic spline interpolation:

$$f\left(\underset{x}{\rightarrow}\right) = a + b *\underset{x}{\rightarrow} + c *\underset{x}{\rightarrow}^2 + d *\underset{x}{\rightarrow}^3 \quad with\ a, b, c, d \in R^3 \qquad \textbf{(Eq. 4.6)}$$

See the next example, Figure 4.34 that shows in a clear way the cubic spline interpolation between the points in the neck of the aneurysm:

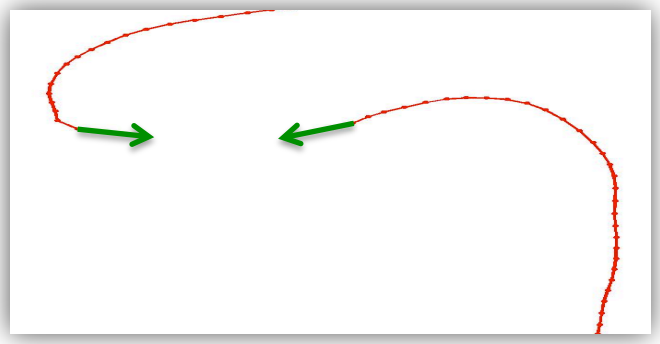


Fig. 4.34 The two green arrows show the direction of the interpolation line between the two points.

The final result is the complete centreline:

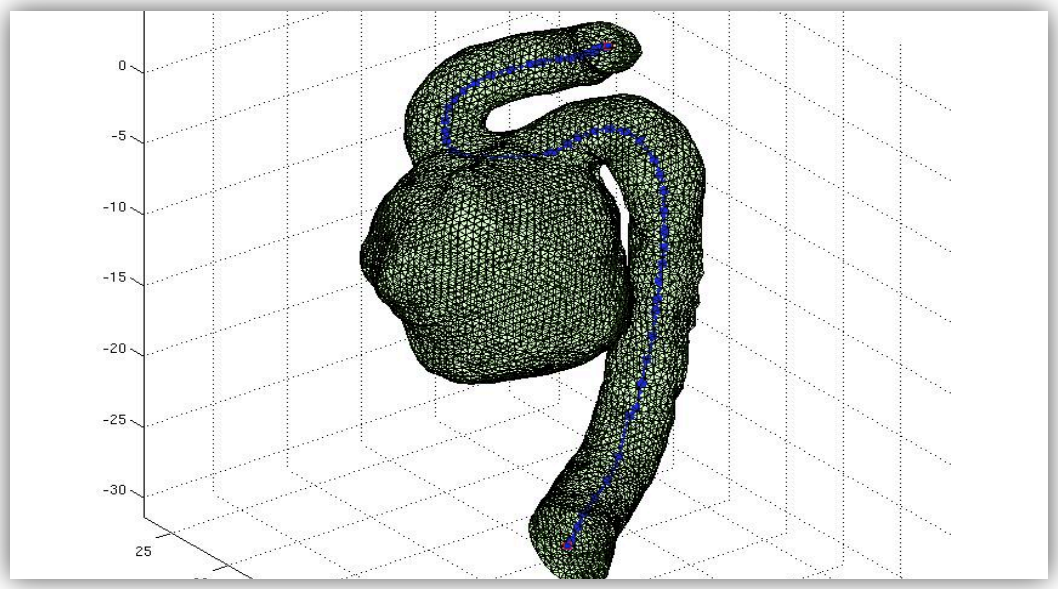


Fig.4.35 Cubic spline interpolation line between two points

Flow Diverter Envelope

- Nominal Coordinate Axis System

After having the CM and the centreline, the coordinate axes system (CAS) is defined. For doing it, a consequent CAS for each point of the centreline has to be chosen. Different options are possible as in the article **[5]** *different methods for a normal orientation in 3D* exposes:

1. Horizontal orientation method: this method fails if the tangent of a point on the curve trajectory is orthogonal to the horizontal plane. Secondly there are two possible orientations for the normal and finally this method is not affine under linear transformations.

2. Principal normal orientation: the problems of this method are, first of all the normal is not defined where the curvature is zero, besides the principal normal orientation turns opposite at the inflection points and the normal can rotate in uncontrolled way around the trajectory curve.

3. Inflection points just in case of planar trajectory curves.

4. PD curves, the main objection to this is the user normally wants to design only the trajectory curve and get the orientation automatically

5. Normal plane projection method: is the one that we choose because it allows to diminish the rotation and it is also insensitive to the inflection points.

As the article said: “choosing an orientation method depends heavily on the shape of a trajectory curve.”

The method chosen define E1, E2 and E3 as:

- E1 is the unit tangent vector to the centreline, pointing in the direction of motion

  $$T = \frac{dr}{ds}$$ **(Eq.4.7)**

- E2 is the derivative of E1 with respect to the arc length parameter of the curve, divided by its length

$$N = \frac{\frac{dT}{ds}}{\|\frac{dT}{ds}\|} \quad \textbf{(Eq.4.8)}$$

- E3 is the cross product of E1 and E2

$$B = T * N \quad \textbf{(Eq.4.9)}$$

The idea is to move the CAS all around the vessel to have always the correct orientation of the axes and also use the E1, the tangent vector of the centreline in a concrete point as the normal vector of a plane.

Figure 4.36 shows the CAS, the red arrow is E1, the green E2 and the blue E3:

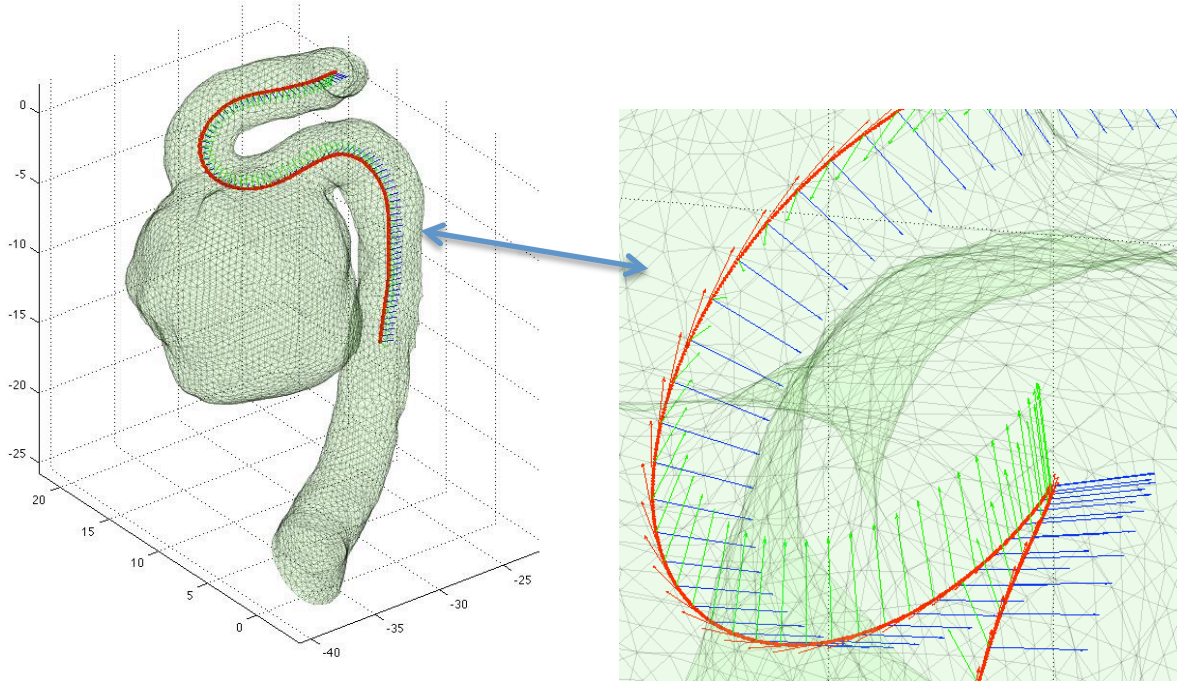


Fig.4.36 CAS and a zoom to see better the arrows

- Creation of the flow diverter envelope

There are different constraints that determine the maximum radius that the FD can take at any point along the centreline, these are called the limitation criteria:

- The radius of the deformed FD cannot be bigger than R_0, its undeformed radius. If it is not the case, a non-existent FD will be produced.
- The radius cannot expand further than R_W, which is the distance to the nearest vessel wall. In other words, the radius has to be lower than the radius dictated by the vessel walls to avoid the FD out of the vessel.
- The radius of the FD also has to satisfy the limitation imposed by the maximum curvature of the centreline (Figure 4.37).

  $R_c = 1/k$ **(Eq. 4.10)**

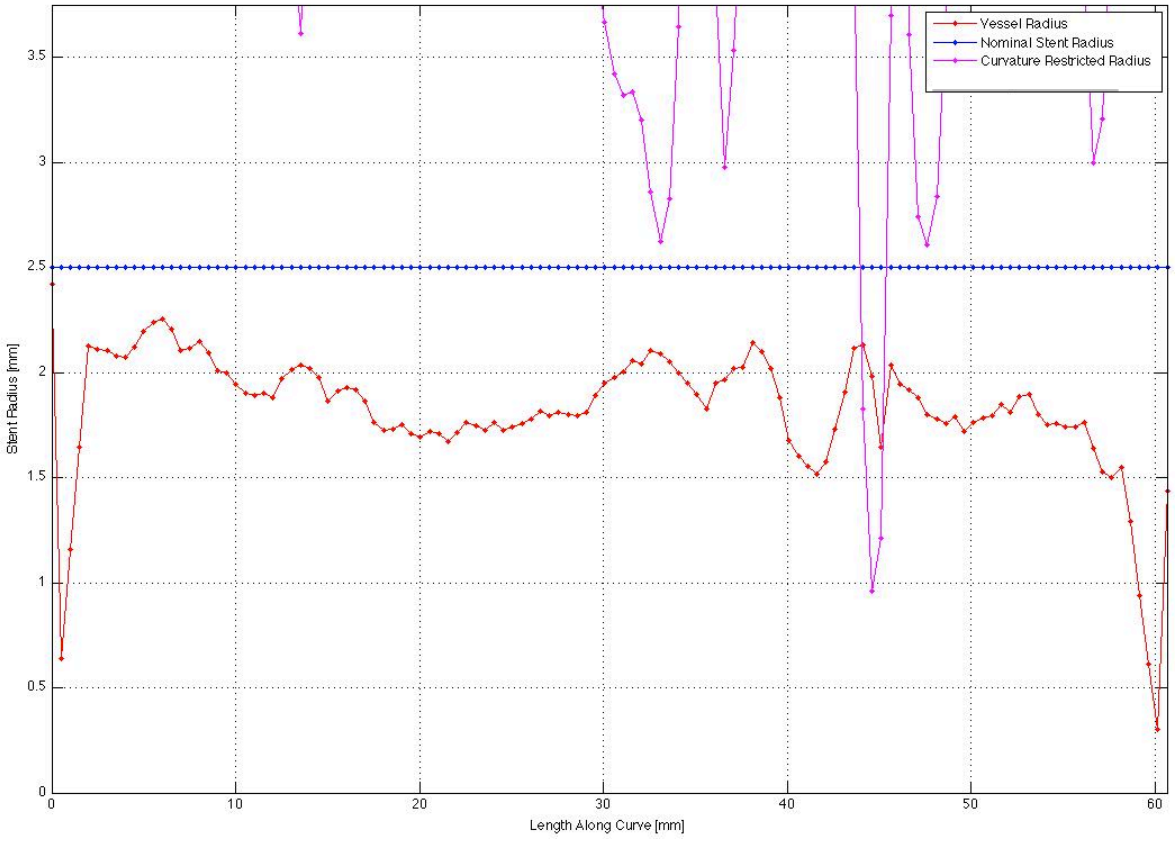


Fig. 4.37 All the radius represented. In the Y-axis: the radius in [mm], in the X-axis the length along the centreline in [mm].

Consequently, the radius chosen at any point along the centreline is the biggest possible radius that still fulfils all the requirements.

Once the centreline has been extracted, the minimum distance between the centreline and the bounding of the vessel is computed in three steps. The finality of finding the distance between the centreline and the bounding of the vessel is to take the minimum distances to then create the virtual flow diverter. This restriction ensures that the flow diverter is within the vessel.

The method can be described as such:

- Define a plane π perpendicular to the centreline. The normal of the plane is E1.
- Search all the triangles that cross the plane π.
- Computation of the distances between the triangles and CM.

In the first step, we go through all the points of the centreline and create the plane π for every point. In Figure 4.38 there is an example, for one point of the centreline, the plane π is created and all the triangles that intersects with the plane π are found.

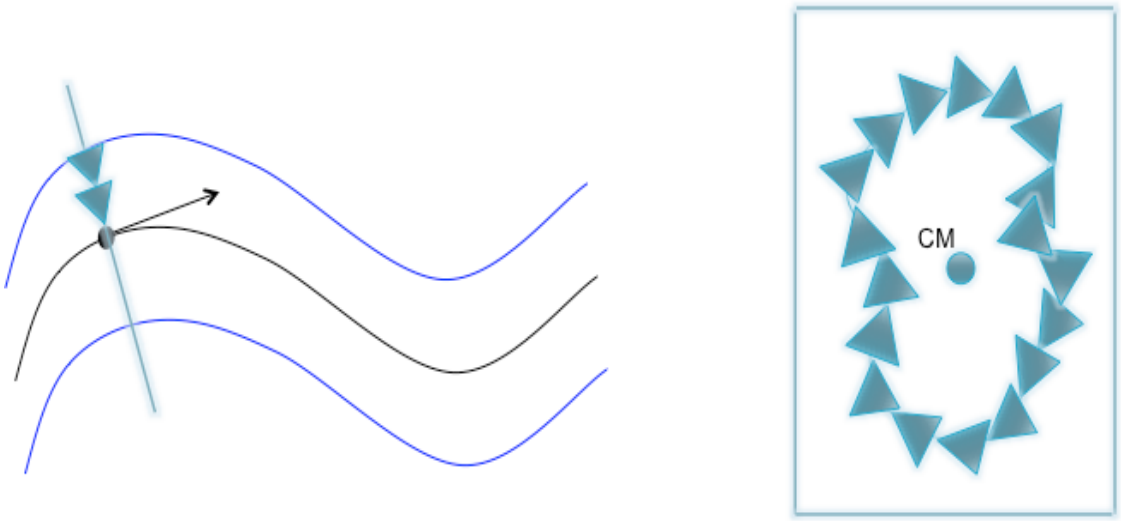


Fig. 4.38 Example of the plane π and the triangles that intersect the plane

Then all the minimum distances (radius) for every slice are taken and represented, just for check if all the circles are inside the vessel.

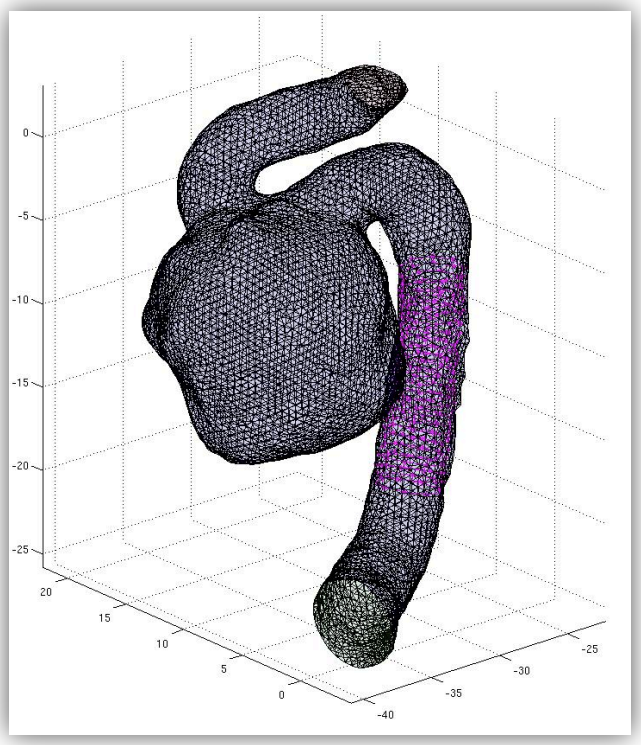


Fig. 4.39 Radius plotted as a circle

The program offers the possibility to include a graphic to smooth the radius function and adapt better the union of the radius to create the flow diverter, but with the actual version the quality of the result is enough so I took it out. (After smoothing the radius function, a new figure appears with the flow diverter plot around the vessel in the tram of the aneurysm).

The first idea is to find the minimum distance between the element and the CM was using the theory solution of the article **[2],** see the code in ANNEX D:

“The problem is to compute the minimum distance between a point P and a triangle T(s,t)=B+sE0+tE1 for (s,t) €D={(s,t):s€[0,1],s+t≤1}. The minimum distance is computed by locating the values (s,t)€D corresponding to the point on the triangle closest to P.”

Using this method we found a problem in the algorithm, the case when the CM is not inside the plane defined by the element (it is not a infinite plane, just the projection of this triangle-plane in one direction, if you see the image in the article, page 2, it is the case out of region “0”) because the point that it finds as the one with the minimum distance it is one point outside of the triangle, and then what this method make it is to take that point and use it as a

centre of one ellipse that fits inside the plane of the triangle. It starts to do the ellipse bigger and bigger until it touches one edge of the triangle being this point the one with the “minimum distance”. That point is not the point that we want as the one with the minimum distance.

A brief test to demonstrate the problem was made and the results are drawn in Figure 4.37.

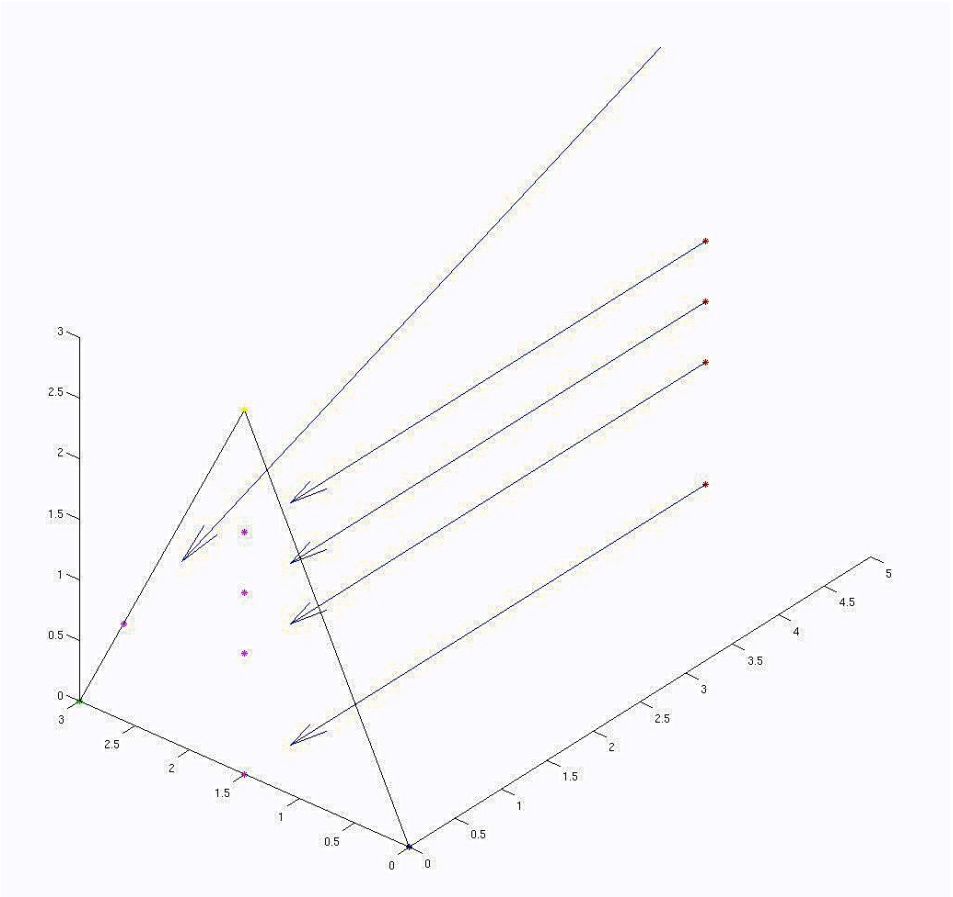


Fig. 4.40 A test with different CM

Finally the procedure used to find the minimum distance is by using cylindrical coordinates to sweep the plane-vessel intersection points with increasing angle and for each consecutive points to build the linear segment connecting them and finds the minimal distance CM-segment. Then the program saves the global minimum to be the output. The points that form the minimum distance are found by the function “GetPointsOnVWall” that uses the function “plane_line_intersect” and then are used into the function “CircumVesselRadFun”.

Produce the Radius Function

In this part we produce the function of the “Radius” to know in any point the radius of the vessel (we use a cubic spline interpolation with some slight smoothing).

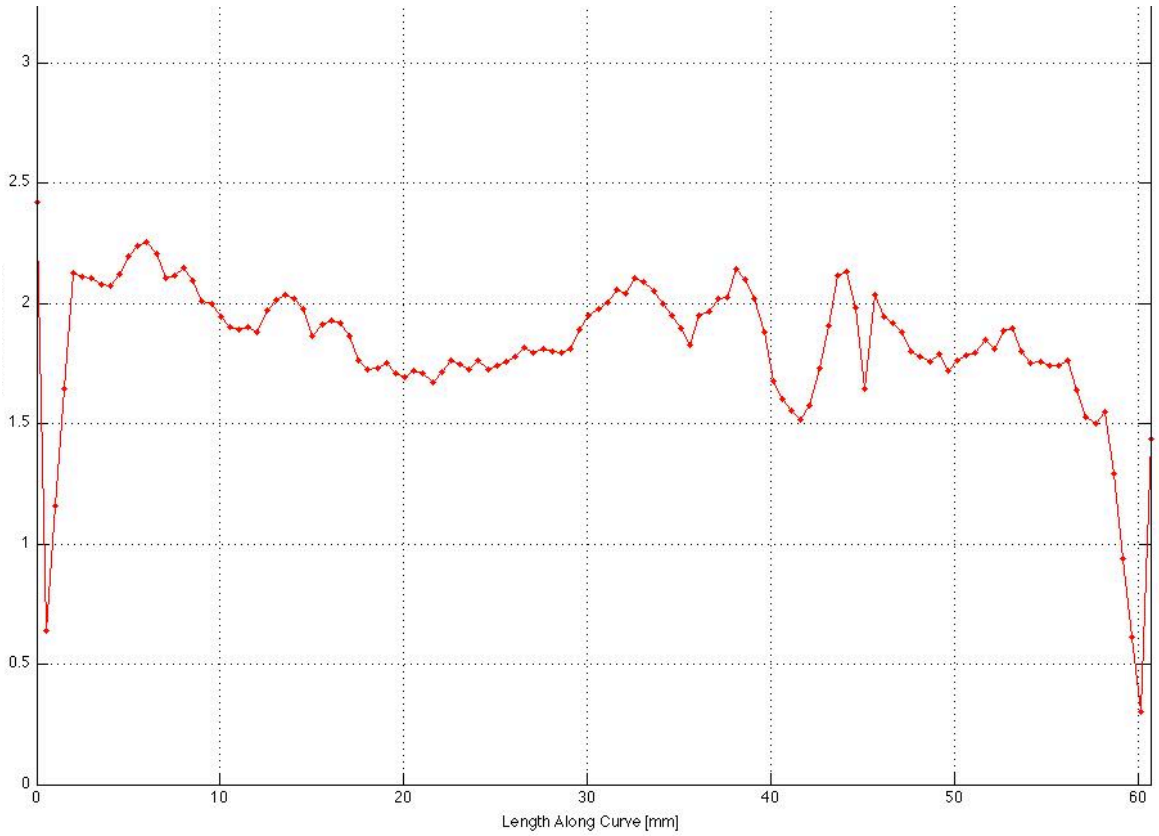


Fig.4.41 Vessel Radius previous to the optimization process. In the Y-axis the radius in [mm] and the length along the centreline in the X-axis in [mm].

Optimisation

Up to now, the program finds the centreline and produces the FD envelope but this procedure seems not to work in the neck of the aneurysm because of the already mentioned impossibility to have a good estimation for the placement of the centreline due to the lack of centre of masses in this part. What in this subchapter is presented is the way to solve the mentioned errors. This part was done in close collaboration with Julien.

Remember that the problems found are:

4.2.2.4.1 To solve the problem of the centreline-CM in the neck of the aneurysm.

4.2.2.4.2 To solve the problem of the limit curvature (k) of the flow diverter.

- To solve the problem of the CM in the neck of the aneurysm

The method used consists to impose a goal radius and then optimize the position of the centreline. This radius is a result of a interpolation between the radius of the last point before the neck of the aneurysm and the first radius of the point after the neck, assuming that in this part the radius of the flow diverter will not change a lot.

The optimization process works due to a cost function that penalizes the part of centreline where the distance centreline-wall is smaller as the goal radius and keeps the curvature of the centreline under the threshold given by stent radius.

It finishes under a stopping condition, when the changes of the position of the centreline falls below a certain threshold. We use a linear interpolation to find the value of the radius in between:

The user has to introduce the values of Param.OptimSlices that refers to the point to start the optimization process and the point to stop.

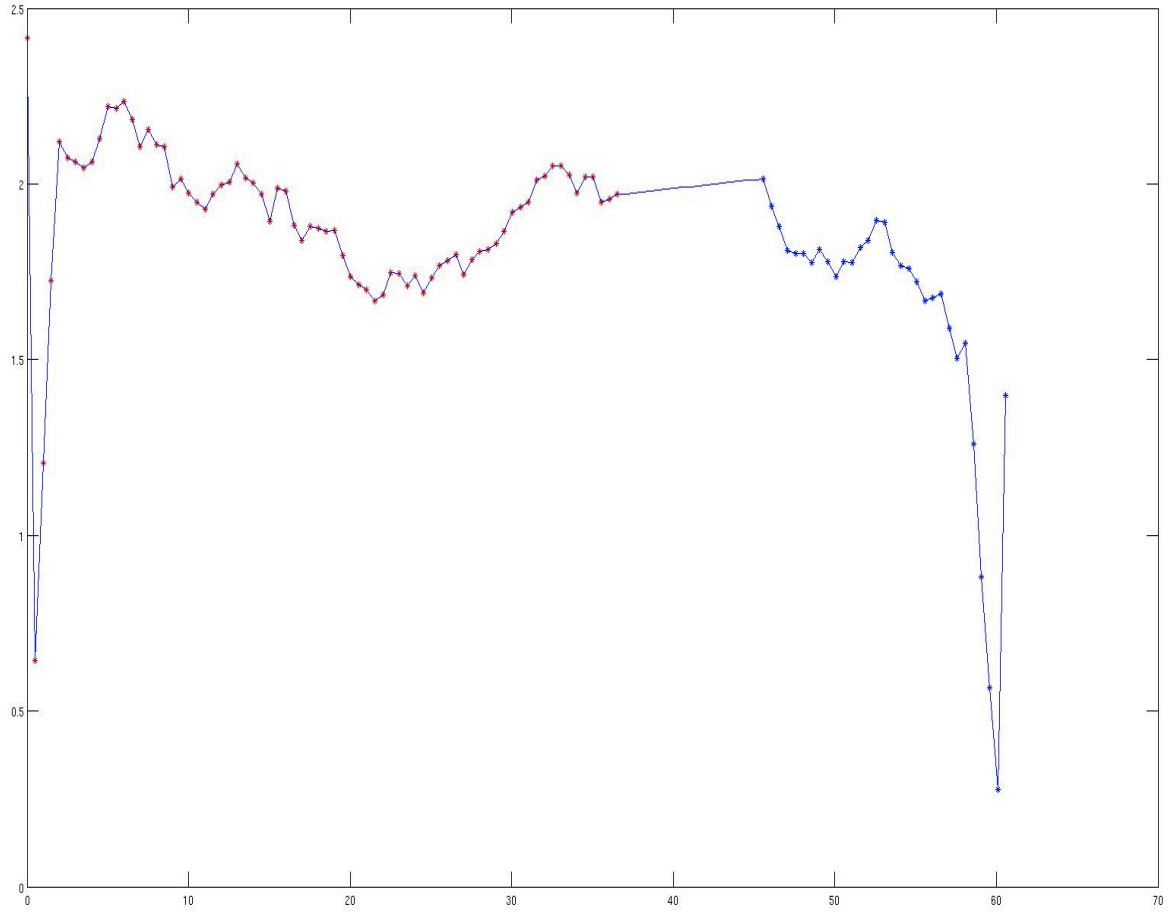


Fig. 4.42 Interpolation radius in the neck of the aneurysm. In the Y-axis the radius in [mm] and the length along the centreline in the X-axis in [mm].

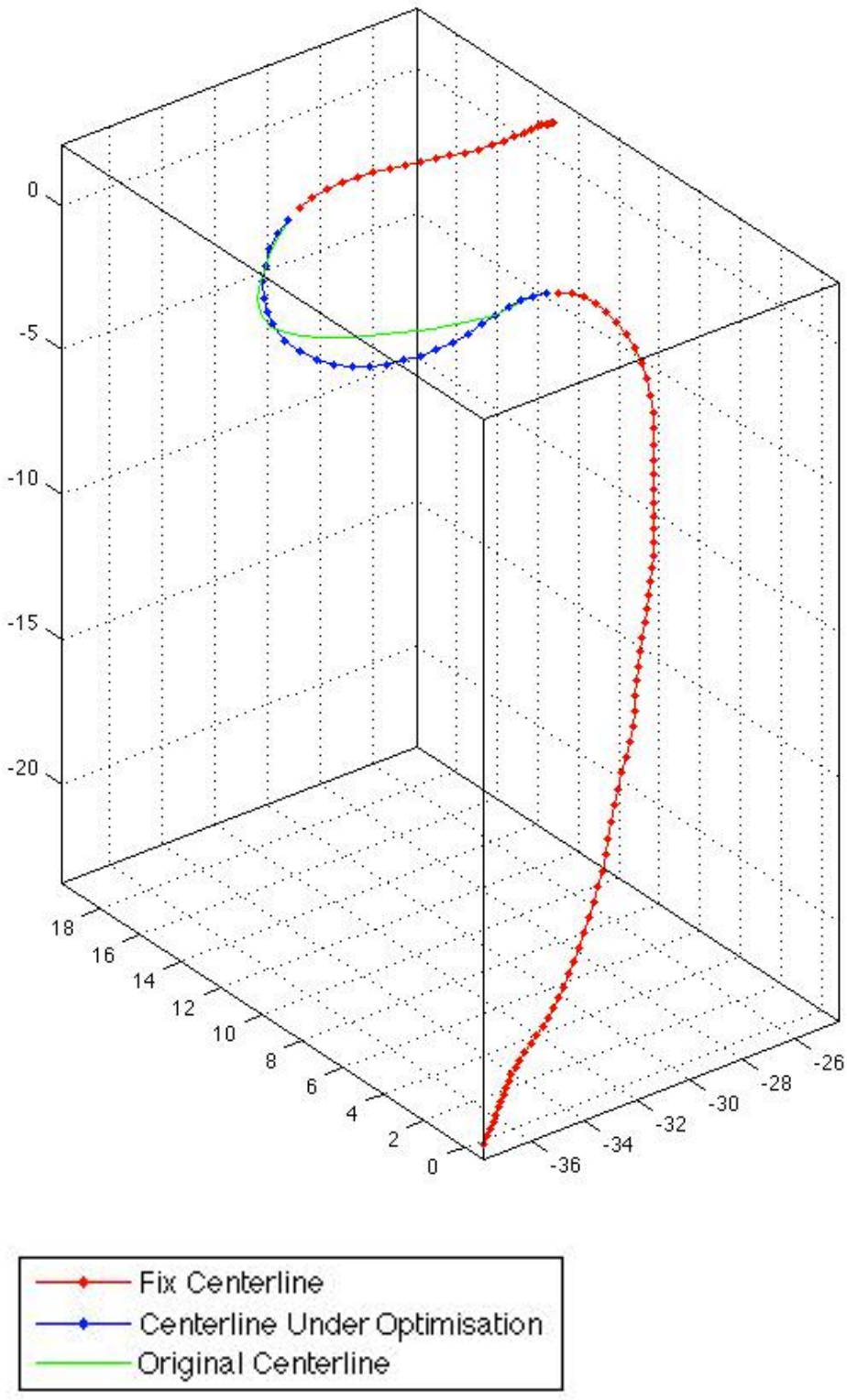


Fig.4.43 Centrelines before and after the optimisation process. [mm]

Figure 4.39 shows the centreline before and after the interpolation process. The new centre of mass is placed finding the new position that diminish the function error.

And with this new radius we find the new CM that contains this radius.

- To solve the problem of the limit curvature of the flow diverter

If you have a real flow diverter in your hands and try to bend it, the maximum bending of the flow diverter is when the struts are touching. So we need to include this constraining to avoid the struts from our virtual flow diverter are crossing (See an example in Figure 4.47 where the coils are touching). Below and Figure 4.44 give in more detail what this paragraph has introduced.

Mathematically, the problem can be described as such:

For every point along the coil r_{coil} (t_i) corresponds an orthogonal projection point on the centreline:

$P(t_i) = r_{Cline}(\tau_i)$

As long as the FD Radius is smaller than the Curvature Radius ($R_{FD}<R_{CLine}$), there exists for every $P(t_1) = r_{CLine}(\tau_1)$ a corresponding $P(t_2 > t_1) = r\ Cline(\tau_2)$ such that $\tau_2 > \tau_1$.

In a simplified way, this means that the coil always “goes forward” with respect to the centreline. Considering one single coil, there are no mechanical objection for a coil to “go backwards”, but given the braided arrangement of all coils, this configuration become then impossible.

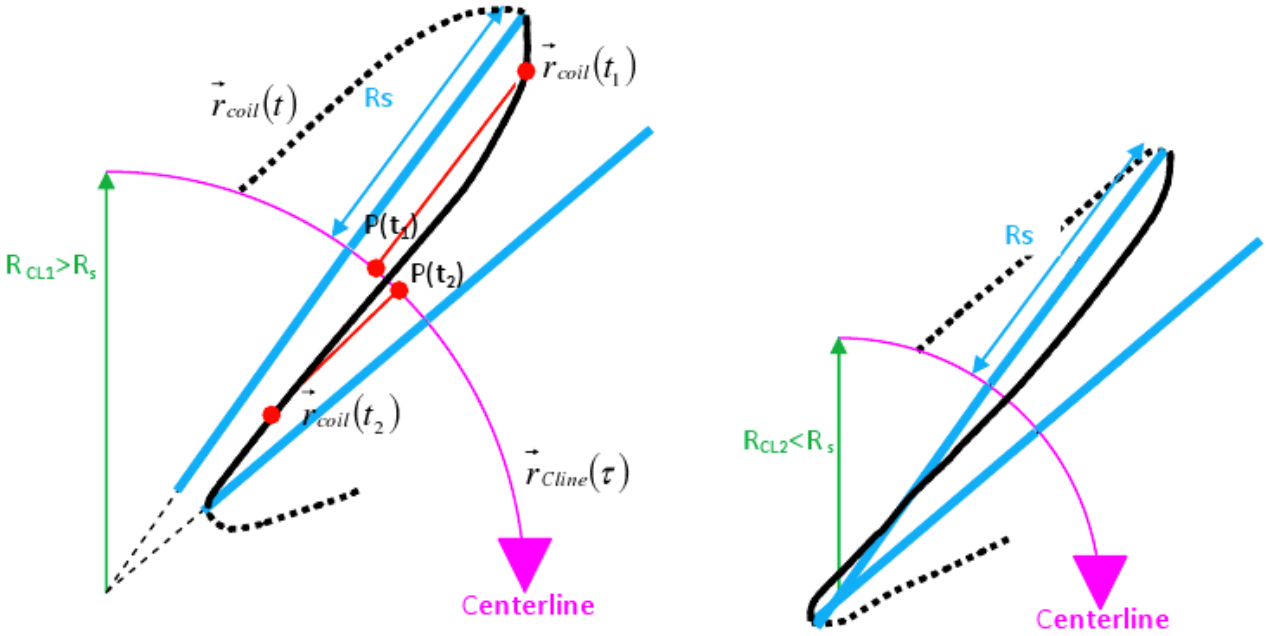


Fig. 4.44 One coil "going forward" and one "going backwards"

To find the limit curvature k an equation is used:

$$k = \frac{1}{R} \qquad \textbf{(Eq. 4.11)}$$

Where R is the radius of the centreline in the point of study.

The equation that I use in the program is:

$$k(i) = sqrt(\left(ddz(i) * dy(i) - ddy(i) * dz(i)\right)^2 + \left(ddx(i) * dz(i) - ddz(i) * dx(i)\right)^2 + \left(ddy(i) * dx(i) - ddx(i) * dy(i)\right)^2)/((dx(i)^2 + dy(i)^2 + dz(i)^2)^{\frac{3}{2}})$$

(Eq. 4.12)

Where "dx" refers to gradient (x) and "ddx" refers to gradient (dx), the same for y and z.

Finally we obtain the virtual flow diverter inserted in the vessel:

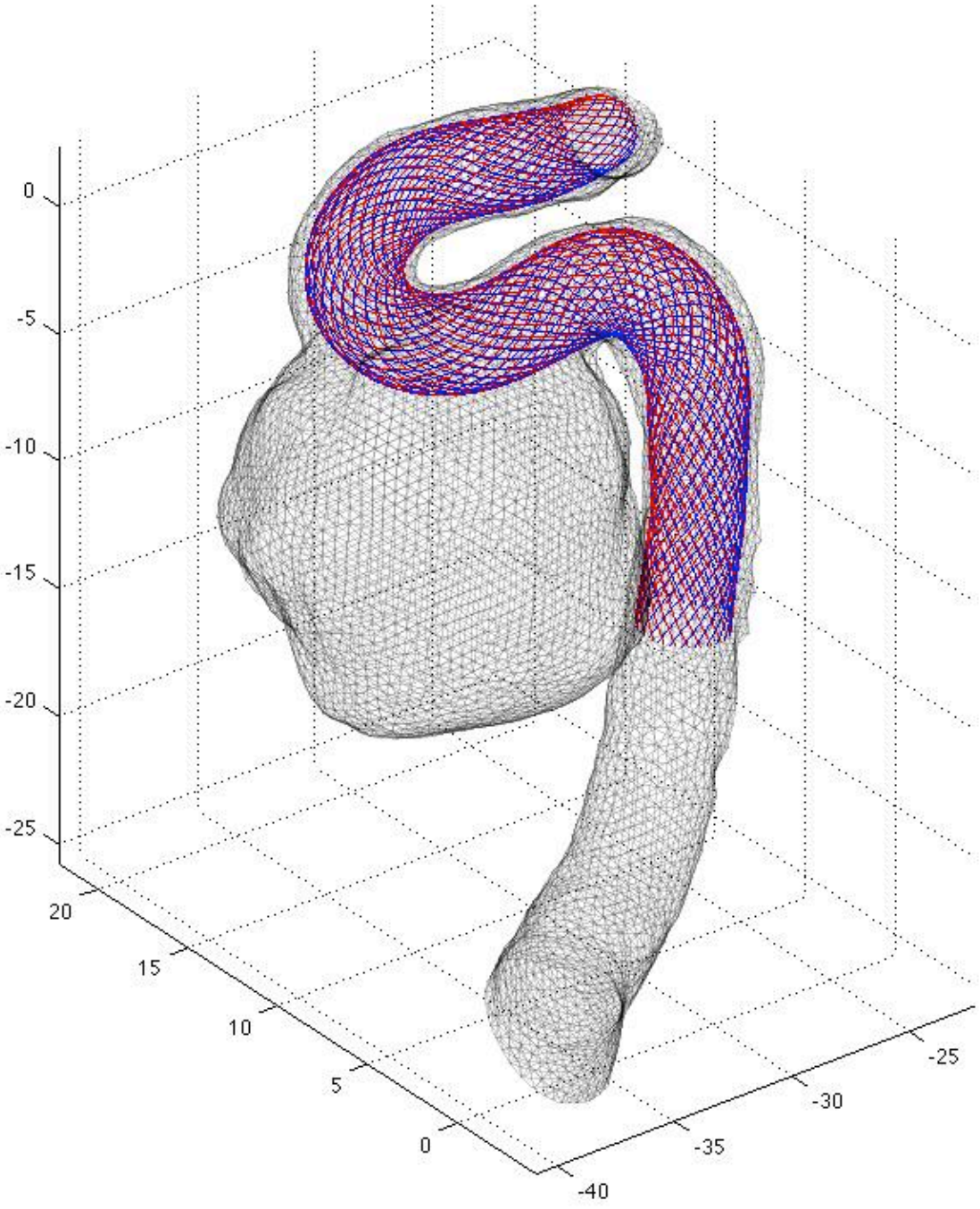


Fig. 4.45 Virtual FD inserted inside the vessel

5.2.3 Output

Thanks of the work we reproduce the virtual flow diverter envelope, the geometric model parameters (not the mechanical model) and we adapted to follow the centreline.

We also produce a STL file of the virtual flow that will be used to make then the different flow simulations.

The production of the flow diverter is all around the vessel, for this reason the user has to play with two parameters to have the FD in the region of interest:

- “Param.tnstart”, when the program starts to produce the virtual flow diverter.
- “Param.tnend”, when the program finishes the production of the virtual flow diverter.

To avoid that the flow diverter touches the vessel, reduce the radius along the line of 0.01mm.

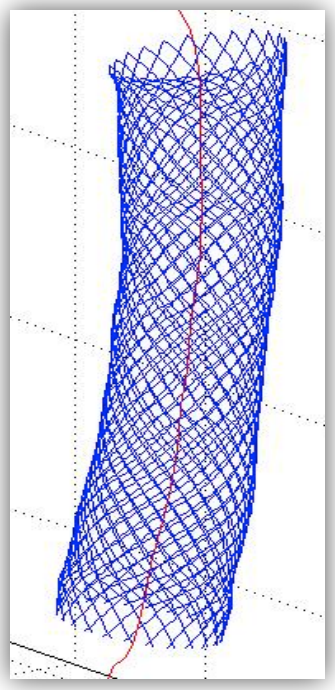


Fig. 4.46 Zoom of the FD

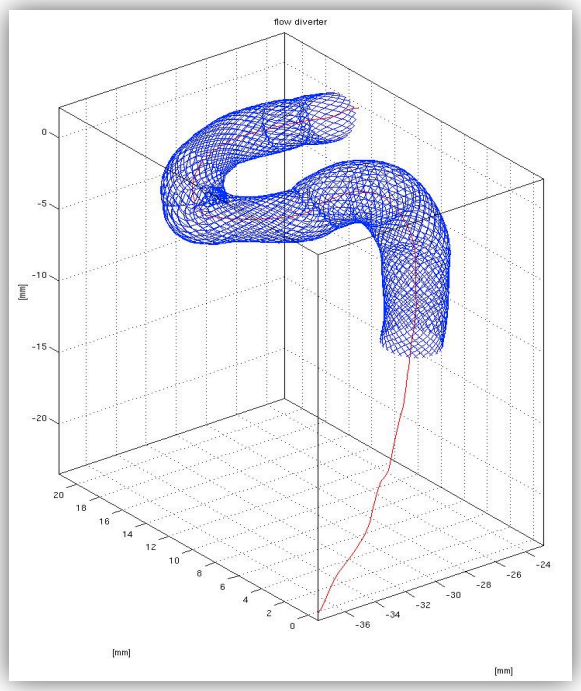


Fig.4.47 Virtual flow diverter after modelling the radius and put it around the vessel before the optimisation process.

See in Figure 4.48 the implementation of the flow diverter inside the vessel:

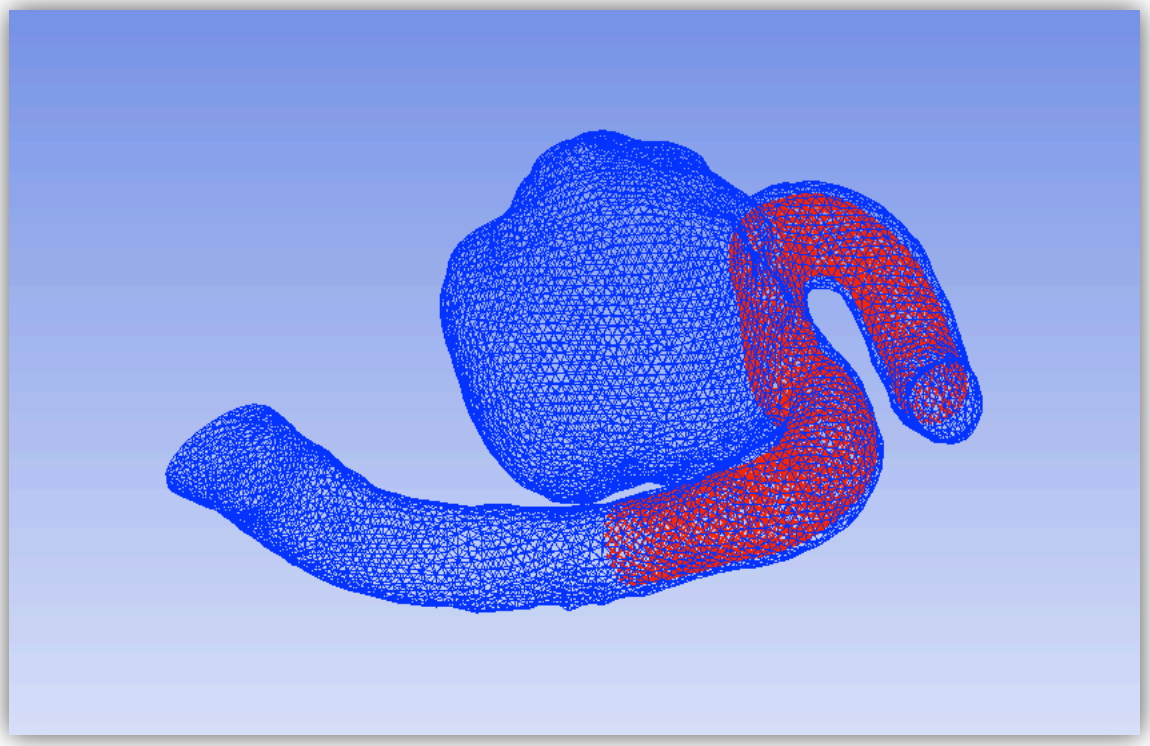


Fig. 4.48 The vessel in blue and the flow diverter in red

The image in Figure 4.47 was before the optimisation process, where the curvature of the vessel makes the struts are crossing among themselves, and having a big changes of the virtual flow diverter shape in the neck of the aneurysm.

Be aware of how it changes after the modifications in Figure 4.45.

6. Results

In this chapter the results of the experimental work are presented. We compare the envelope of the virtual flow diverter obtained with the methodology described in Chapter 4 against a real diverter inserted into a patient.

The reason of comparing these two data is because this is the only example that we have with the same patient before and after the flow diverter implementation.

The comparisons are made by taking the two flow diverter envelopes:

- The envelope of the virtual flow diverter
- The envelope from the implemented flow diverter

The goal is to make now another segmentation and extract the second envelope to validation. The reason why we compare the envelopes and not the position of each individual strut is because if we take a look to our CT data, the quality of the image is not good enough to extract just the flow diverter (the struts).

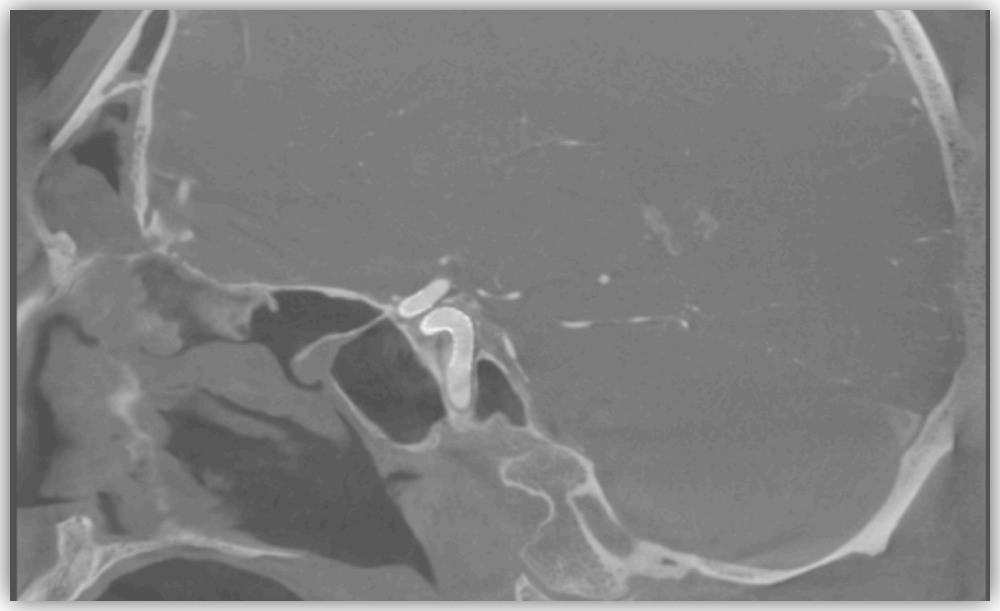


Fig. 5.1 One slice of the CT where we can see the FD

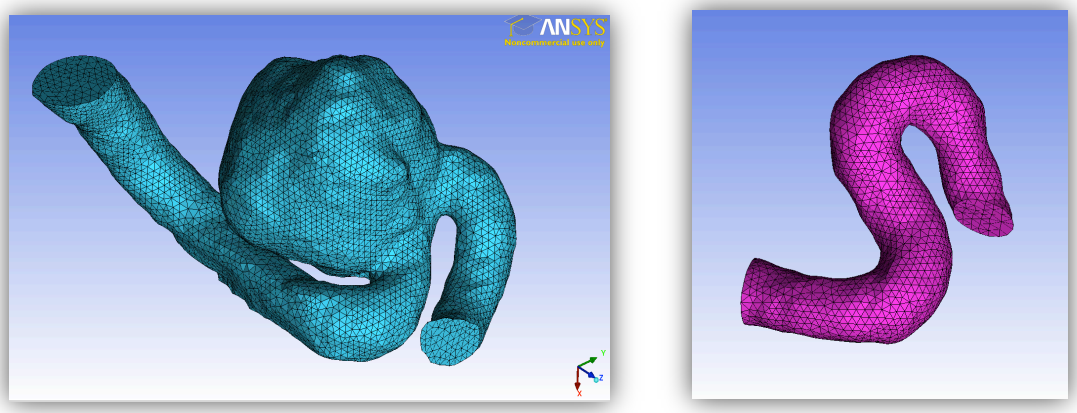


Fig. 5.2 The two vessels from we extract the envelopes to compare the virtual flow diverter (left image) and the real one (right image).

Summary of the different tests that are made in this chapter:

- To compare the distance between the two centrelines
- To compare the Radius
- To compare the ovality

6.1 To compare the distance between the two centrelines

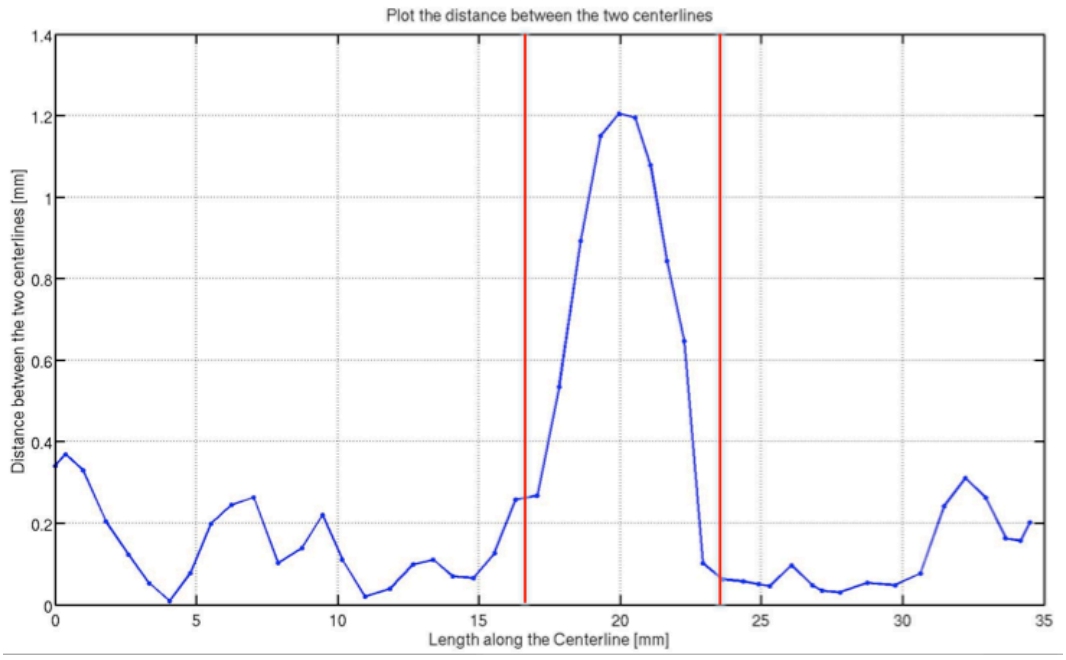


Fig. 5.3 Distance between the two centrelines

In the Y-axis the distance between the two centrelines in [mm] and the length along the centreline in the X-axis in [mm]. The two vertical lines delimit the points that belong to the neck of the aneurysm.

This first test is a comparison of the position of the two centrelines.

As the Figure 5.3 shows, the error between the centrelines is lower than 0.4 mm except in the part between the two red straight-vertical lines, delimit the points that are part of the neck of the aneurysm. This error is logic because the distance to interpolate between the two centres of mass is too long, for this reason the optimization in this part was necessary.

The Y-axis is the distance between the two centrelines for every point tested in [mm] and the X-axis the length along the centreline [mm].

6.2 To compare between the radiuses

We made the assumption that the virtual stent envelope has always a circular shape (radius = R). The comparison was between this R and the minimum radius and maxim radius of the real flow diverter envelope.

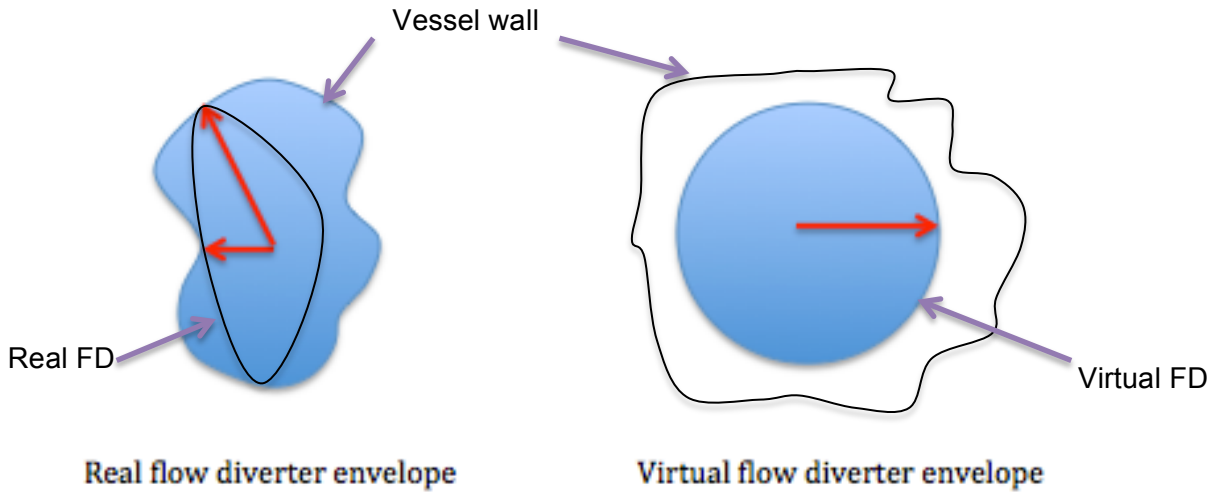


Fig.5.4 Two different envelops, the first one is a random shape

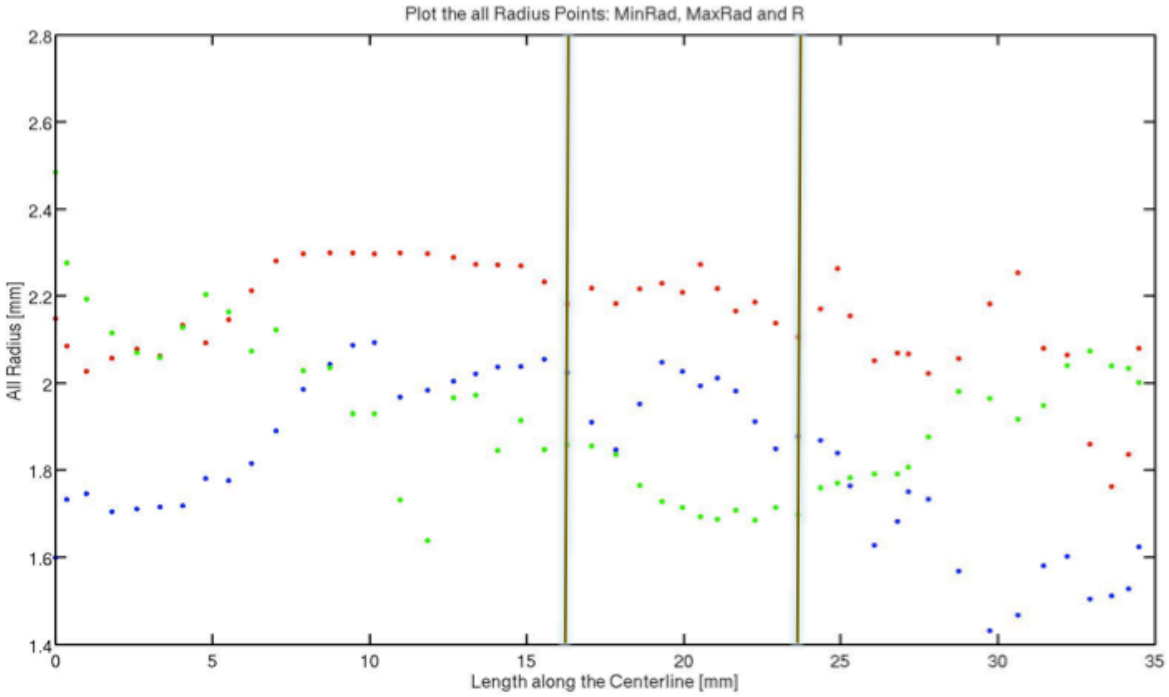


Fig. 5.5 Plotting the value of the radius of the CM's. Y-axis All Radius [mm], X-axis Length along the centreline [mm]. The two vertical lines delimit the points that belong to the neck of the aneurysm.

The idea of plotting Figure 5.5 is to use it as a visual check to look the different radius:

- Red points: "MaxRad" is the maximum distance between the CM and the bounding of the vessel.

- Blue points: "MinRad" is the minimum distance between the CM and the bounding of the vessel.

- Green points: R

The minimum radius of the real flow diverter envelope is between 1.7 and 2.1 mm and its maximum radius is between 2 and 2.3 mm. The difference between the radius are not too big, we can't take the maximum distance to produce the virtual flow diverter because in that case we'll have some parts of the virtual flow diverter sticking out.

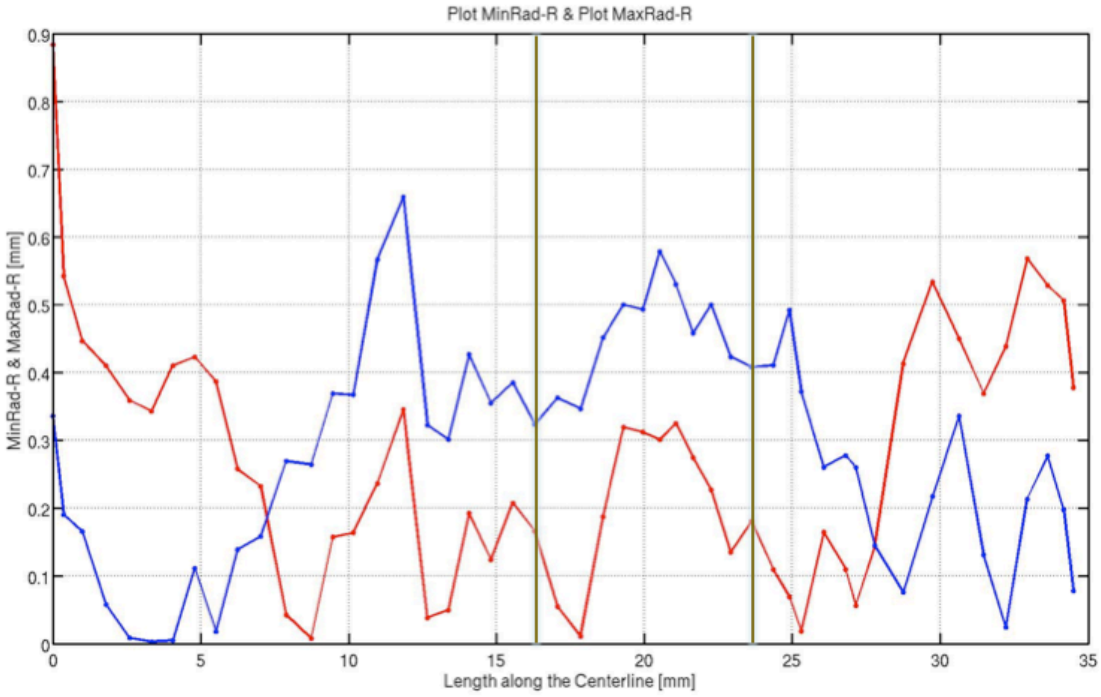


Fig. 5.6 Difference between the different radius MinRad-R (red line) and MaxRad-R (blue line). The two vertical lines delimit the points that belong to the neck of the aneurysm.

The Y-axis refers to the MinRad-R & MaxRad-R. It is in [mm] and the X-axis, it is the length along the centreline in [mm].

The red line is the: $Radius\ difference\ 1 = MinRad - R$ **(Eq.5.1)**

The blue line is the: $Radius\ difference\ 2 = MaxRad - R$ **(Eq.5.2)**

6.3 To compare the ovality

The functions that are used to study the ovality between the two envelopes are:

$$Radius\ Ratio(1) = \frac{MinRad}{R}$$ **(Eq.5.3)**

$$Radius\ Ratio(2) = \frac{MaxRad}{R}$$ **(Eq.5.4)**

$$Ovality = \frac{MaxRad}{MinRad}$$ **(Eq. 5.5)**

Equation 5.3 shows how much R differs from MinRad.

Equation 5.4 shows how much R differs from MaxRad.

Equation 5.5 shows how oval the real stent is.

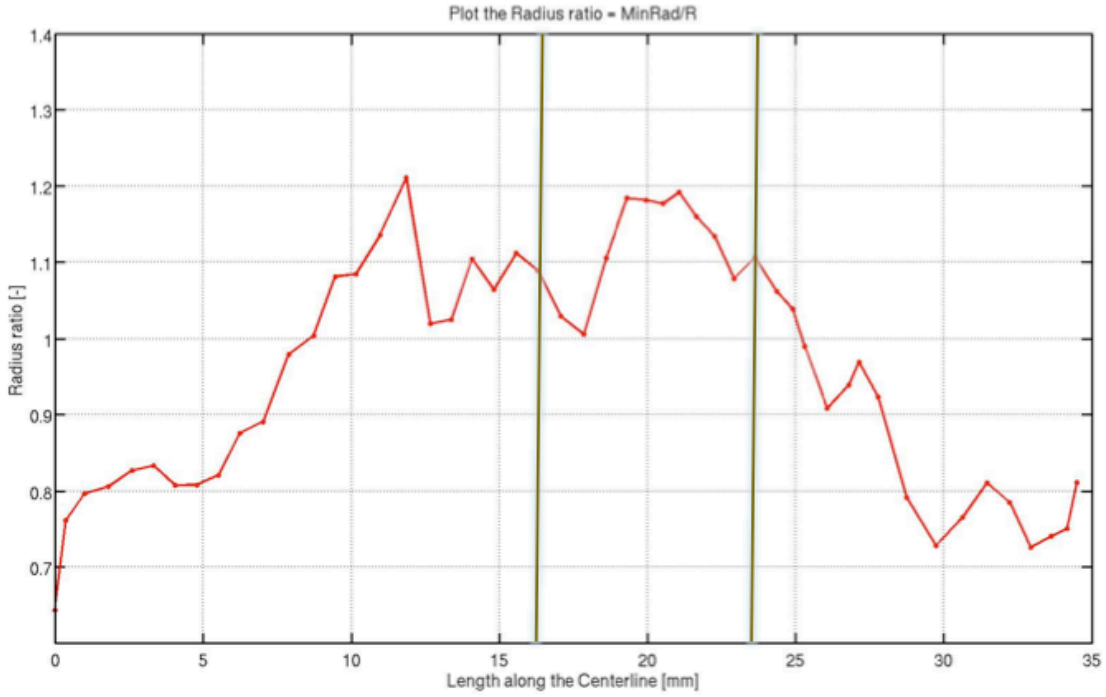


Fig. 5.7 Radius Ratio (1). The Y-axis is the radius ratio(1) [-] and the X-axis refers to the length along the centreline in [mm]. The two vertical lines delimit the points that belong to the neck of the aneurysm.

The ideal case is to be around the value one, these would means that the values are the same. If we took all the values in average are closer to one.

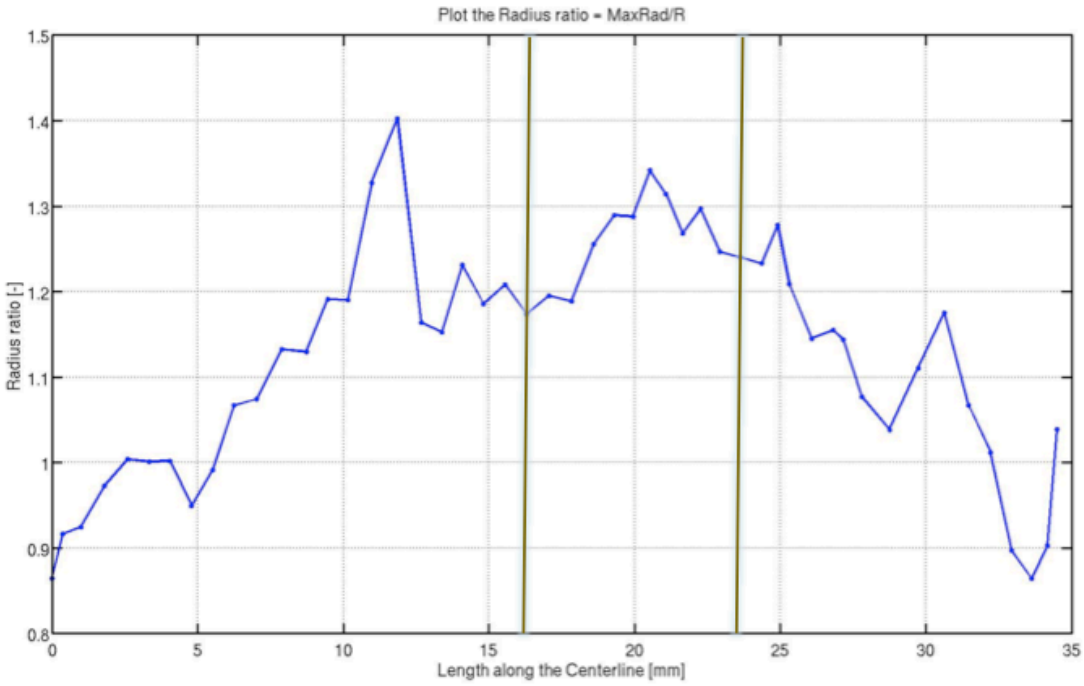


Fig. 5.8 Radius Ratio(2), Y-axis [-], X-axis length along the centreline in [mm]. The two vertical lines delimit the points that belong to the neck of the aneurysm.

Using the MaxRad instead of MinRad the error is bigger, between 1 and 1.4 [-].

In the two tests explained before a systematic error (systematic offset of the real size of the vessel) is observed. What you can see in the CT is not the real shape of the vessel, because the material of the coils produces a distortion in the image looking bigger than what in reality is.

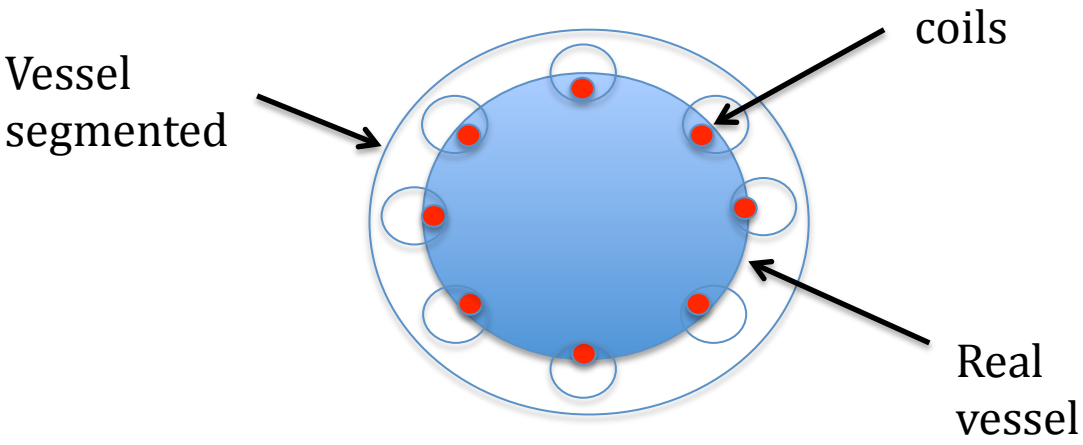


Fig. 5.9 Distortion of the CT

This causes a systematic offset because we took a bigger envelope that what in really it is, but we can work with the results that we obtained. This is what Figure 5.9 tries to reproduce. In blue we can see the normal size of the vessel, in red the real size of the coils, the small circles in blue are what the CT reproduces as the coils distortion because the metallic material of it looks bigger than in reality and finally the big circle is what the CT shows us.

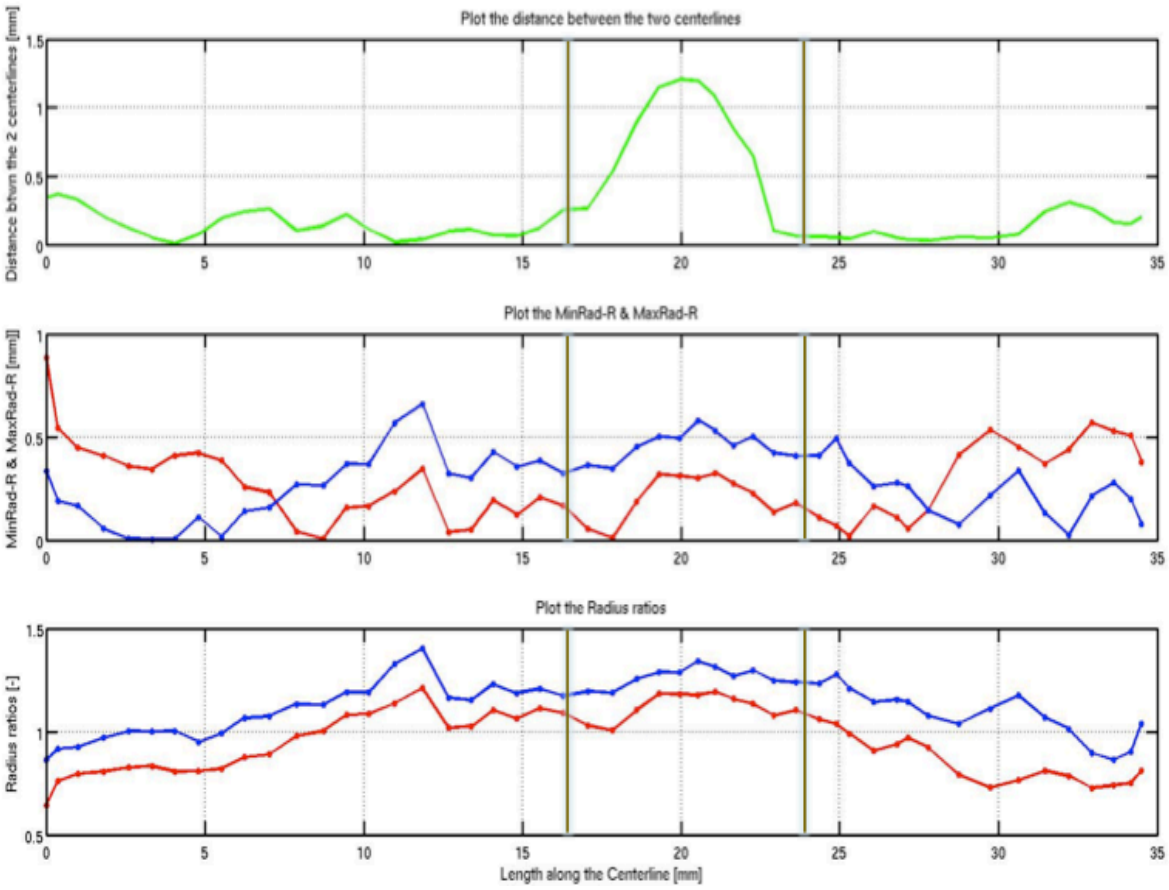


Fig. 5.10 Summary of the different results obtained before.

In Figure 5.10, the first graphic represents the plot of the distance between the two centrelines in [mm], the second one the differences between the Radius in [mm] and the third one the Radius ratio [-]. The three graphics have the same X-axis, length along the centreline in [mm]. The two vertical lines in every graphic delimit the points that belong to the neck of the aneurysm.

7. Cost of the project

This is a scientific project oriented in the research field for this reason when we think about the costs of the project it is necessary to differentiate between the hours spent in:

- Formation, it consists in learning how to use the programs needed in the project and increase my knowledge in an unknown field for me. I consider that the time spent for my formation has not to be included in the calculation of the costs because in a future I can use this knowledge in a future project.
- Design and implementation, it refers to the time dedicated to create, to develop and to execute the program → The cost per hour for an engineer is: 50 €/hour
    - $Total\ hours = 17 weeks * 5 \frac{days}{week} * 8 \frac{hours}{day} = 680\ hours$
    - $Total\ Cost\ Design\&Implementation = 50 \frac{€}{hour} * 680\ hours = 34000€$
- Writing, it is the time spent writing the thesis.
    - $Total\ hours = 4 weeks * 5\ \frac{days}{week} * 8 \frac{hours}{day} = 160\ hours$
    - $Total\ Cost\ Writing = 50 \frac{€}{hour} * 160\ hours = 8000\ €$

Since the project has been developed at a university lab, it is difficult to estimate the infrastructure costs associated to the occupation of a desk. The aim of this chapter is only to estimate the costs in staff, hardware and software. Referring to the costs of hardware and software:

- Hardware cost:
    - CPU 920, processor Intel ® Core™ i7, 8GB of RAM, 500 GB hard disk: 2500CHF
    - Graphic card NVIDIA GeForce GT 240: 72 €
    - The amortization of a computer is 3 years.
    - $Cost\ of\ the\ PC = 2050\ € * \frac{1}{3 years} * \frac{1 year}{12 months} * 5 months = 284.72€$

- Software cost:
    - MATLAB license: 123 €
    - Image Processing Toolbox: 370 €
    - Duration of the license: 1 year
    - $Cost\ of\ MATLAB = 123\ € + 370\ € = 493€$

The cost of the MATLAB license and the Image Processing Toolbox is what the university ETH from Zurich pays.

$$Total\ Cost\ Equipment = 284.72€ + 493€ + 72€ = 849.72\ €$$

Finally, I need to include the cost of printing the report:

- Printing expenses: 120€

This price includes the printing of the project in colour (2 copies of the project), 5 CD's, the tags and the binding of the project.

TOTAL COST OF THE PROJECT:

$$Total\ Cost = Total\ Cost\ Equipment + Total\ Cost\ Report\ + Total\ cost\ Writing$$
$$+ Total\ Cost\ Design\&Implementation = 42969.72\ €$$

8. Social & Environmental Impact

The project has a low environmental impact. Since it has been realized in an office and only using a computer, its main impact is the energy consumption, which is particularly high because of the use of the graphic card to perform the calculations in parallel. The computer needs a more powerful source alimentation (650W) than a normal computer. At the end of the lifespan of the graphic card, it needs to be carried to a green point or a dump because it is not recyclable.

Concerning the social impact, the project does not have a direct social impact. However, the research line in which it is framed can at a medium and long term, have a positive impact on health, and therefore on social welfare.

Conclusions

The methods used to develop this thesis such as the way to find the centreline and the implementation of the virtual flow diverter inside the artery are one of the multiple possibilities that nowadays exist but if we take into account the knowledge learned in my home university (ETSEIB, UPC) and the materials that I had in the ETH I obtained a program easy to use, fast to execute and with good results.

In my opinion one of the advantages using MATLAB is the facility that the user has to make different tests and act interactively with the program to obtain different results. So now a new good step is to improve the program efficiency and implement a complementary procedure to optimize the centreline in the neck of the aneurysm.

I also suggest finding another method to segment the data from the CT because using a manual method could produce different results depending the person who is making the segmentation.

Besides, in the future during the flow simulations maybe we notice that we need to introduce more parameter in the production of the virtual Flow Diverter or completely change its morphology depending on the results obtained by the PhD student with the flow simulations.

The part that takes more time is the obtention of the STL file so another recommendation is to find another way to export the FD to ANSYS CFX.

Finally, I would recommend to try to optimize the selection/detection of the points to interpolate in the neck of the aneurysm because now it is done manually introducing the number of the points in “Parameters.m”. A nice solution could be to select directly the points from the screen.

To conclude say that some of these recommendations are already being taken into account.

Acknowledgements

I would like to thank Professor Gabor Székely for having given me the possibility to do this project of the Computer Vision Laboratory from ETH Zentrum in Zürich, Switzerland and Doctor Sven Hirsch the supervisor of Julien and me who was always willing me to collaborate, he shows me the different parts of the project and he introduced me to the all team.

Julien thank you very much for helping me in everything, for being patient with me, for explaining very well the different steps to follow and the things that I did not understand. I learned a lot working with you.

Thanks to:

- Doctor Med. Zsolt Kulcsar, specialist for Radiology and Neuroradiology, Klinik Hirslanden Zürich.

- Professor Toni Susin who made me a super-introduction in the use of MATLAB and helped me to pull forward the first few days, sorry to bother you so much...

-The tutor of my thesis Professor Dani Tost, for helping me in the project.

- Professor Pere Caminal who recommended me the department where I made the project also my mentor (an expert on this field) and helped me deciding about my future.

Thanks a lot to my parents because without them it would not have been possible for me to go to Zurich and my sister for helping me in different things.

Cristina who was at my side at all times.

And finally to all the friends I made in Zurich, thanks to them I had six unforgettable months!

Bibliography

Bibliographic references

[1] BRADFORD J.WOOD, M.D. and POUNEH RAZAVI, M.D. Virtual Endoscopy: A promising New Technology. , National Institute of Health, Bethesda, Maryland. Am Fam Physician.2002,Jul,1;66(1):107-113

[2] DAVID EBERLY, Distance Between Point and Triangle in 3D. Geometric Tools, LLC, September 28, 1999 http://www.geometrictools.com/

[3] FREDERIC F.LEYMARIE and BENJAMIN B.KIMIA, The Shock Scaffold for Representing 3D Shape. Brown University, Division of Engineering, Providence RI, USA, May2001

[4] NICU D.CORNEA, DEBORAH SILVER, PATRICK MIN, Curve-Skeleton Properties, Applications and Algorithms.

[5] PEKKA SILTANEN AND CHARLES WOODWARD, Normal orientation methods for 3D offset curves, sweep surfaces and skinning. Helsinki University of Technology, Department of Computer Science, Otakaari 1 A, SF-02150 Espoo Finland. Eurographics '92, volume 11 (1992), number 3

[6] RICHARD A.ROBB, PH.D., Virtual (Computed) Endoscopy: Development and Evaluation using the visible human datasets. National library of medicine, National Institute of Health, Bethesda, Maryland. Mayo Fundation/Clinic

[7] SUNIL APPANABOYINA, FERNANDO MUT, RAINALD LÖHNER, CHRISTOPHER PUTMAN, JUAN CEBRAL, Simulation of intracranial aneurysm stenting: Techniques and challenges. Comput. Methods Appl. Mech. Engrg. 198(2009) 3567-3582

[8] VARIOUS WRITERS, Standardized evaluation methodology and reference database for evaluating coronary artery centreline extraction algorithms. Medical Image Analysis 2009

[9] http://www.brainaneurysm.com/

Complementary bibliography

[CB1] http://www.nlm.nih.gov/medlineplus/aneurysms.html

[CB2] http://espanol.ninds.nih.gov/trastornos/Aneurismas_Cerebrales.htm

[CB3] http://www.emedicinehealth.com/aneurysm_brain/article_em.htm#Brain_Aneurysm Causes

[CB4] http://www.paraview.org/

[CB5]Images%20Power%20Point/images%20from%20the%20net/TAAF%20%20Brain%20Aneurysms%20%20Treatment.html

[CB6] http://www.slicer.org/

[CB7] http://www.slicer.org/slicerWiki/index.php/Slicer3.4:Training

[CB8] http://www.ansys.com/Products/Other+Products/ANSYS+ICEM+CFD

[CB9] http://www.nlm.nih.gov/medlineplus/spanish/ency/article/001122.htm

[CB10] http://www.mathworks.com/

[CB11] http://www.ennex.com/~fabbers/StL.asp

[CB12] http://en.wikipedia.org/wiki/Curvature

[CB13] http://en.wikipedia.org/wiki/Frenet%E2%80%93Serret_formulas

[CB14] http://www.angelfire.com/fl/houseofbartlett/solutions/line2tri.html

[CB15] http://www.microvention.com/Products/Coils/tabid/62/Default.aspx

[CB16] Daniel H¨ahn, Coronary Artery Centerline Extraction in 3D Slicer using VMTK based Tools, diploma thesis, Ruperto Carola University of Heidelberg, Germany, February 2010.

[CB17] David Arnold, Curvature in Matlab. Math 50C — Multivariable Calculus.

[CB18] ELEONORA FLORE, IGNACIO LARRABIDE, LORENZA PETRINI, GIANCARLO PENNATI, ALEJANDRO FRANGI, Stent deployment in aneurysmatic cerebral vessels: Assessment and quantification of the differences between Fast Virtual Stenting and Finite Element Analysis. INRIA-00418412, version 1-18 Sep 2009

[CB19] Luca Augsburger1, 2, Mohamed Farhat3, Philippe Reymond1, Edouard Fonck1, 2, Zsolt Kulcsar2,Nikos Stergiopulos1, Daniel A. Rüfenacht2, Effect of Flow Diverter Porosity on Intraaneurysmal Blood Flow.

[CB20] Nicu D. Cornea, Deborah Silver, Rutgers University, New Jersey, USA and Patrick Min, American University of Rome, Italy, Curve-Skeleton Properties, Applications and Algorithms

[CB21] Sonia Pujol, Ph.D., Interactive Editor Tutorial. Surgical Planning Laboratory, Harvard Medical School.

[CB22] Sky McKinley and Megan Levine Cubic Spline Interpolation. Math 45: Linear Algebra.

[CB23] Sven Loncaricand Dubravko Cosic and Atam P. Dhawan, Segmentation of CT Heads. Faculty of Electrical and Computer Engineering, University of Zagreb Unska 3, 10000 Zagreb, Croatia. Department of Electrical and Computer Engineering, University of Cincinnati, Cincinnati, OH 45221, USA.

[CB24] Toni Susín, Simulación de Partículas, Laboratorio de Simulación Dinámica. Dept. Matemàtica Aplicada 1 (UPC) http://www-ma1.upc.es/~susin

Annexes

ANNEX A: MATLAB CODES

ParameterFile.m……98

Main.m……100

CircumVesselRadFun.m……102

CoMWithMapping.m……106

ControlFlags.m……109

CreateTheCenterline.m……110

Do1DCoil.m……111

DoCoilCicles.m……113

DoCoilElipses.m……113

DoIntraXing.m……113

DoMesh.m……115

DoNoXingSegments……123

DoStlFiles.m……126

DoTheCoilPts.m……127

DoTheStentMesh.m……131

DoXing.m……133

DoXingSegments.m……135

Ellypse.m……140

ErrorFun.m……141

FindNewCenterOfMass.m……142

FlagFace.m……145

FlagLines.m……147

FlagPtsBetwWalls.m……148

GetCoM.m……151

GetCurvature.m……153

GetInitCurvature.m……154

GetNominalCAS.m……155

GetPointsOnVWall.m……157

GetRadInterpolation.m……159

GetStentCAS.m……161

GlobalRadDiff.m……164

InPolyedron.m……165

LengthOfcscvn.m……170

plane_line_intersect.m……171

PosDependentCurvature.m……172

PosDependentRadius.m……174

ProduceNotOptRadFun.m……175

ProduceRadFun.m……176

ProduceRadFunNew.m……178

SelectMeshTri.m……180

SelectMeshTriRect.m……182

SelectOneTri.m……185

SmoothMidLine.m……188

Stlread.m……..191

TimeOfcscvn.m……194

ParameterFile.m

```
disp('Parameter File...')

%% Names
%--------------------------------------------------------------------------
global Name;

Name.StlFolder_Path = '/Users/david_fs/Desktop/matlab versio
12_05_2011/StentMeshingDV4';
Name.StlFile_Name   = 'meshstent+aneurysmultversio';
Name.StlFile_Ext    = '.stl';

Name.StlFile_Path   = fullfile(Name.StlFolder_Path,...
    [Name.StlFile_Name,Name.StlFile_Ext]);


% Parameters
global Param
Param.AngleForFace = 10;
Param.COMDistX   = 2; % mm
Param.COMDistY   = 0.2; % mm
Param.COMDistZ   = 0.2; % mm

Param.SliceDist      = 0.5; % mm
Param.OptimSlices    = (72:89);
Param.OptimPtsSelect = 0;
Param.ROffset        = 0.15;

Param.LengthOfcscvnTol = 10^-3;
Param.TimeOfcscvnTol   = 10^-3;
Param.DistToNextXTol   = 10^-3;

% Display Control
Scrsz                       = get(0,'Screensize');
Param.FigPos                = [100,100,Scrsz(3)-200,Scrsz(4)-200];
Param.WaitTimeout           = 1;
% Param.NrOfPtsForCLineInterp = 10;
Param.MaxMovdx = 1;% mm
Param.MaxMovdy = 1;% mm

% Stent Properties
global Stent
Stent.StartDist = 20;  %0.4; %1;  % From the face1 in mm
Stent.CoilNrOf  = 24; % Per direction
Stent.NOfTurns  = 2; % -

Stent.R0        = 2.5; % mm
Stent.Length0   = 23; % mm
Stent.PPitch0   = Stent.Length0/(Stent.NOfTurns*2*pi);
Stent.CRad      = 0.04; %mm

% Optimisation Parameters
Param.OptimisationFlag       = 0;
Param.DoMeshFlag             = 0;
Param.OptimVisualisationFlag = 1;
```

```
Param.PreOptimRunNrOf        = 2;

% Smoothing Parameters
Param.NPtsArndSmtg = 10;  % Number of not optimised point used in the
smoothing
Param.SmtgCoef     = 0.4; % Smoothing coefficient
Param.SmtgPolModel = 2;   % Polynomial Degree for the Savitzky-Golay
smoothing filter.
```

Main.m

```
clear all;
close all; clc;

% Initialisation
%-------------------------------------------------------------------------
% Calling Global Values
global Name
global F; global V; global N;
global NomCAS;

% Input File
ParameterFile

% Load the vessel
[F,V,N] = Stlread(Name.StlFile_Path);

%% ------------------------------------------------------------------------
%% 1    COM And MidLineFun
% 1.1 Flagging
% 1.2 Partial Vessel Segments
% 1.3 Create the Centerline
%-------------------------------------------------------------------------
%% 1.1 Flagging
GlobalFaceFlag = FlagFace();
GlobalLineFlag = FlagLines(GlobalFaceFlag);
ControlFlags(GlobalFaceFlag,GlobalLineFlag,1)

%% 1.2 Partial Vessel Segments
[CoM,BBFlags] = GetCoM(GlobalFaceFlag,1);

%% 1.3 Create the Centerline and the frenet axis
CLineFun = CreateTheCenterline(CoM,1);

%% ------------------------------------------------------------------------
%% 2    Nominal Stent Envelope
% 2.1 Nominal Frenet CAS (Coordinate Axis System)
% 2.2 Circumferential Vessel Wall Fun
% 2.3 Produce the Radius function
%-------------------------------------------------------------------------
%% 2.1 Nominal CAS (Coordinate Axis System)
GetNominalCAS(CLineFun,1)

%% 2.2 Circumferential Vessel Wall Fun
GetPointsOnVWall(GlobalFaceFlag,0);
CircumVesselRadFun(0) % Still needs improvement to automatise the
procedure

%% 2.3 Produce the Radius function
% Takes in to account:
% - The constraint du to curvature of midline
% - The contraint because of the wall
[RadFun,Rc] = ProduceRadFun(1);
```

```
%% ------------------------------------------------------------------------
%% Section 3 (Optional):
% 3.1 Optimisation
%-------------------------------------------------------------------------
if Param.OptimisationFlag == 1
    GoalRValues         = GetRadInterpolation(RadFun,1);      %radius
neck aneurysm
    [NewCLPts,RadOptim] = FindNewCenterOfMass(GoalRValues,1); %CoM neck
aneurysm
    CLineFun            = SmoothMidLine(NewCLPts);
    Rc                  = GetCurvature(CLineFun);
    RadFun              = ProduceRadFunNew(RadOptim,Rc,GoalRValues,1);
    GetStentCAS(CLineFun,RadFun,'(after Optimisation)',1)
else
    RadFun              = ProduceNotOptRadFun(Rc,1);
    CLPts               = NomCAS.Origin;
    CLineFun            = SmoothMidLine(CLPts);
    GetStentCAS(CLineFun,RadFun,'a',1)
end
%% ------------------------------------------------------------------------
%% Section 4:
% 4.1  Dist Dependent CAS
% 4.2B Manual Radius Function Tunning
% 4.3  Get Coils Intersections
% 4.4  Equally Divide Coil Segments
% 4.5  Produce Mesh
%-------------------------------------------------------------------------
Do1DCoil(RadFun,CLineFun,1)
if  Param.DoMeshFlag == 1
    DoMesh()
end
```

CircumVesselRadFun.m

```
function CircumVesselRadFun(PlotFlag)
% close all;
% PlotFlag = 0; clc;
disp('CircumVesselRadFun...')

%% Global Variables
%--------------------------------------------------------------------------
global NomCAS
global Name
global Param
global VessPts
global CVRFun

%% See if the variables exists
%--------------------------------------------------------------------------
if exist([Name.StlFile_Name,'_6_CircumVessFun.mat'],'file')
    TmpNomCAS =
load([Name.StlFile_Name,'_6_CircumVessFun.mat'],'CVRFun');

    if isfield(TmpNomCAS.CVRFun,'VessMinRad') %&&...
%             isfield(TmpNomCAS.CVRFun,'VessMaxRad') &&...
%             isfield(TmpNomCAS.CVRFun,'VessAngleMinRad') &&...
%             isfield(TmpNomCAS.CVRFun,'VessAngleMaxRad')

         CVRFun = TmpNomCAS.CVRFun;

        disp(['=> "Min/Max Radius variables" from ',Name.StlFile_Name,...
            '_6_CircumVessFun.mat have been loaded'])
        return
    end
end

%% Local Variables
%--------------------------------------------------------------------------
SlicesToPlot = 1:size(NomCAS.Origin,1);
% SlicesToPlot = [12,13,14];

%% Local variables
%--------------------------------------------------------------------------
AllRadMin      = NaN*ones(1,size(NomCAS.Origin,1));
AllRadMax      = NaN*ones(1,size(NomCAS.Origin,1));
AllAngleRadMin = NaN*ones(1,size(NomCAS.Origin,1));
AllAngleRadMax = NaN*ones(1,size(NomCAS.Origin,1));
for SliceNr = 1:size(NomCAS.Origin,1)
% for SliceNr = 87
    if PlotFlag == 1 && ismember(SliceNr,SlicesToPlot)
        figure('Name',['SliceNr: ',num2str(SliceNr)])
    end


    %% Do the Transformation
    %--------------------------------------------------------------------------
```

```
% Functions if needed
% TrafVec =@(Vec) (TrafoMtx*Vec')';
% TrafPts =@(Pts) (TrafoMtx*(Pts-repmat(Orig,[size(Pts,1),1]))')';
Orig = NomCAS.Origin(SliceNr,:);
TrafoMtx = [NomCAS.E1(SliceNr,:)/norm(NomCAS.E1(SliceNr,:));...
            NomCAS.E2(SliceNr,:)/norm(NomCAS.E2(SliceNr,:));...
            NomCAS.E3(SliceNr,:)/norm(NomCAS.E3(SliceNr,:))];
Pts3D  = (TrafoMtx*...
    (VessPts(SliceNr).Coord-...
    repmat(Orig,[size(VessPts(SliceNr).Coord,1),1]))')';
Pts2D   = [Pts3D(:,2),Pts3D(:,3)];

% Eliminate points that are too close to each other
[b,m,n] = unique(roundn(Pts2D,-10),'rows');
Pts2D   = Pts2D(m,:)';

% Choose the points inside a polygon
if Param.OptimPtsSelect==1 && ismember(SliceNr,Param.OptimSlices)
    Fig = figure('Position',Param.FigPos,'Name','Choose Points',...
        'NumberTitle','off');
    plot(Pts2D(1,:), Pts2D(2,:),'.b')
    hold on
    plot(0,0,'*r')
    xlim([min(Pts2D(1,:))-0.1*(max(Pts2D(1,:))-min(Pts2D(1,:))),...
          max(Pts2D(1,:))+0.1*(max(Pts2D(1,:))-min(Pts2D(1,:)))])
    ylim([min(Pts2D(2,:))-0.1*(max(Pts2D(2,:))-min(Pts2D(2,:))),...
          max(Pts2D(2,:))+0.1*(max(Pts2D(2,:))-min(Pts2D(2,:)))])
    hold on
    axis equal;
    h  = impoly;
    xyPoly = getPosition(h);
    In = inpolygon(Pts2D(1,:), Pts2D(2,:),xyPoly(:,1),xyPoly(:,2));
    SelPts = find(In);
    Pts2D = Pts2D(:,SelPts);
    plot(Pts2D(1,:), Pts2D(2,:),'or');
    drawnow
    uiwait(Fig,1)
    close(Fig)
end

%% Reorder the points with increasing angle
%-------------------------------------------------------------------------
% Compute the angles of the given points
apt = mod(atan2(Pts2D(2,:),Pts2D(1,:))+2*pi,2*pi);
% Reorder Everything & add the "last point" = "first point"
axy = [apt;Pts2D(1,:);Pts2D(2,:)];
axy = sortrows(axy')';
axy(:,end+1) = [2*pi+axy(1,1);axy(2,1);axy(3,1)];

%% Loop the points
%-------------------------------------------------------------------------
% Initialisation
GlobalRmin = Inf; Globalamin = NaN;
GlobalRmax = 0;   Globalamax = NaN;
for i=1:size(Pts2D,2)
    x1 = axy(2,i);
    x2 = axy(2,i+1);
    y1 = axy(3,i);
```

```
        y2 = axy(3,i+1);
 % Find the equation of the line f[x]= aa * x + bb
        % With: f[x1]==y1 && f[x2]==y2
        aa(i) =(y1-y2)/(x1-x2);
        bb(i) =(x1*y2-x2*y1)/(x1-x2);
        % Same Line in PolarCoord
        % With xpol =@(R,alpha) R.*cos(alpha) && ypol =@(R,alpha)
R.*sin(alpha);
        % in f[x]= y =  a * x + b
        rr =@(alpha) repmat(bb(i),[1,length(alpha)])./...
            (sin(alpha)-aa(i)*cos(alpha));
        alphamin(i) = axy(1,i);
        alphamax(i) = axy(1,i+1);
        % Find the minimum and maximum radius
        [amin,Rmin]    = fminbnd(@(x) rr(x),alphamin(i),alphamax(i));
        [amax,InvRmax] = fminbnd(@(x) 1/rr(x),alphamin(i),alphamax(i));
        Rmax           = InvRmax.^-1;
        % Register the min and max
        if Rmin<GlobalRmin
            GlobalRmin = Rmin;
            Globalamin = amin;
        end
        if Rmax>GlobalRmax
            GlobalRmax = Rmax;
            Globalamax = amax;
        End

% Plot
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        if PlotFlag == 1 && ismember(SliceNr,SlicesToPlot)
            % The points
            plot(Pts2D(1,:),Pts2D(2,:),'.r')
            hold on
            % The center
            plot(0,0,'*r')
            % Prepare the data of the cartesian line
            ff =@(x) aa(i)*x+bb(i);
            xx = linspace(x1,x2,2); yy = ff(xx);
            plot(xx,yy,'--r');
            % Polar to cartesian transformation
            alphas = linspace(alphamin(i),alphamax(i),10);
            RR     = rr(alphas);
            xpol   =@(R,alpha) R.*cos(alpha);
            ypol   =@(R,alpha) R.*sin(alpha);
            plot(xpol(RR,alphas),ypol(RR,alphas),'--b')
            % Plot the considered limit point
            plot([x1,x2],[y1,y2],'om')
            % Local min and max
%             plot([0,Rmin*cos(amin)],[0,Rmin*sin(amin)],'--r')
%             plot([0,Rmax*cos(amax)],[0,Rmax*sin(amax)],'--b')
            axis equal; grid on
        end
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    end
    AllRadMin(SliceNr)      = GlobalRmin;
    AllRadMax(SliceNr)      = GlobalRmax;
    AllAngleRadMin(SliceNr) = Globalamin;
    AllAngleRadMax(SliceNr) = Globalamax;
```

```
    VessPts2D(SliceNr).Coord = Pts2D;
% Plot
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    if PlotFlag == 1 && ismember(SliceNr,SlicesToPlot)
plot([0,GlobalRmin*cos(Globalamin)],[0,GlobalRmin*sin(Globalamin)],...
            '-m','Linewidth',2.5)
plot([0,GlobalRmax*cos(Globalamax)],[0,GlobalRmax*sin(Globalamax)],...
            '-k','Linewidth',2.5)
    end
end
CVRFun.VessMinRad      = AllRadMin;
CVRFun.VessPts2D      = VessPts2D;
save([Name.StlFile_Name,'_6_CircumVessFun.mat'],'CVRFun');
```

CoMWithMapping.m

```
function CMBackTransfo =
CoMWithMapping(F,V,N,BBFlag,GlobalFaceFlag,WallPts)
disp('CoMWithMapping...')
%% Global Values
%--------------------------------------------------------------------------
global Param
%% Internal Flags
%--------------------------------------------------------------------------
PlotThings = 0;
%% Coordinate Transformations
%--------------------------------------------------------------------------
YDir(1,1:3) = WallPts(4,1,:)-WallPts(1,1,:);
Dpth(1,1:3) = WallPts(1,2,:)-WallPts(1,1,:);
XDir(1,1:3) = cross(YDir,Dpth);
V12(1,1:3)  = WallPts(2,1,:)-WallPts(1,1,:);
if dot(V12/norm(V12),XDir/norm(XDir))<0
    XDir = -XDir;
end
ZDir(1,1:3) = cross(XDir,YDir);
TrafoMtx = [XDir/norm(XDir);YDir/norm(YDir);ZDir/norm(ZDir)];
VV    = (TrafoMtx*V')';
NN    = (TrafoMtx*N')';
FrontPt1(1,1:3) = WallPts(1,1,:);
FrontPt2(1,1:3) = WallPts(2,1,:);
XOrig = (TrafoMtx*FrontPt1')';
XTip  = (TrafoMtx*FrontPt2')';
FrntWallPts(:,1:3)=WallPts(:,1,1:3);
BackWallPts(:,1:3)=WallPts(:,2,1:3);
WallPtsT(:,1,:) = (TrafoMtx*FrntWallPts')';
WallPtsT(:,2,:) = (TrafoMtx*BackWallPts')';
%% Plot the vessel
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
if PlotThings==1
    LocFig  = figure('Position',Param.FigPos);
    LocSurf = trisurf(F,VV(:,1),VV(:,2),VV(:,3),0);
%     plot3(VV(:,1),VV(:,2),VV(:,3),'.b')
    grid on; axis equal
end
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~




%% Calculat the centers of Mass
%--------------------------------------------------------------------------
--

% Bounding box
bboxPts = [min(VV,[],1);max(VV,[],1)];
XMin = XOrig(1)    ; XMax = XTip(1);
YMin = bboxPts(1,2); YMax = bboxPts(2,2);
ZMin = bboxPts(1,3); ZMax = bboxPts(2,3);

% (XMax-XMin)     Param.SliceDistX
NrOfSlicX = ceil(abs((XMax-XMin)/Param.COMDistX));
```

```
NrOfSlicY = ceil(abs((YMax-YMin)/Param.COMDistY));
NrOfSlicZ = ceil(abs((ZMax-ZMin)/Param.COMDistZ));
% Subdivision Of The Space Box (box world):
XLen = XTip(1)-XOrig(1);
if NrOfSlicX==1
    xVec = 0.5*(XOrig(1)+(0.03*XLen)+XTip(1)-(0.03*XLen));
else
    xVec = linspace(XOrig(1)+(0.03*XLen),XTip(1)-(0.03*XLen),NrOfSlicX);
end
yVec = linspace(bboxPts(1,2),bboxPts(2,2),NrOfSlicY);
zVec = linspace(bboxPts(1,3),bboxPts(2,3),NrOfSlicZ);
[xMat,yMat,zMat] = meshgrid(xVec,yVec,zVec);

% points inside this box
T  = [xMat(:),yMat(:),zMat(:)];

% Detect points inside a manifold closed surface:
inRow = InPolyedron(VV,F,NN,T);

% Points inside the vessel in increasing x-value
TT = sortrows(T(inRow,:),1);

% Condition: The X-Slice Cannot cross One Face
%--------------------------------------------
[Tb,ndif]=unique(TT(:,1)); % ndif: indices in which there is change
'slice'
SliceInterFaceFlag = zeros(length(Tb),1);

% Check if the X-Plane crosses the face
% FaceNrOf = length(unique(GlobalFaceFlag))-1;
% for i=1:length(Tb)
%     for j=1:FaceNrOf
%         BeforPlaneFlag = VV(F(find(GlobalFaceFlag==j)),1)<Tb(i);
%         AfterPlaneFlag = VV(F(find(GlobalFaceFlag==j)),1)>Tb(i);
%         if sum(BeforPlaneFlag)>0 && sum(AfterPlaneFlag)>0
%             SliceInterFaceFlag(i)=1;
%         end
%     end
% end

% Calculation of CoM
%-------------------
ndif = [1;ndif]; % add the first point
cntr = 0;
CoMWaitbarH = waitbar(0,'Calculation position of CoM...');
for i = 1:length(Tb)
     waitbar(i/length(Tb),CoMWaitbarH);
    if SliceInterFaceFlag(i)==0
        cntr = cntr + 1;
        AllPtsOnSlice = TT(ndif(i):ndif(i+1),:);
        % Check if the points are inside the wall (if necessary)
        %-------------------------------------------------------
        PtsForCMFlag = FlagPtsBetwWalls(AllPtsOnSlice,WallPtsT);
        % Plot the all the points on the slice
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        if PlotThings == 1
            hold on
```

```
            plot3(AllPtsOnSlice(:,1),AllPtsOnSlice(:,2),...
                  AllPtsOnSlice(:,3),'.r');
            plot3(AllPtsOnSlice(find(PtsForCMFlag==1),1),...
                  AllPtsOnSlice(find(PtsForCMFlag==1),2),...
                  AllPtsOnSlice(find(PtsForCMFlag==1),3),'*m');
        end
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        CM(cntr,:) = mean(AllPtsOnSlice(find(PtsForCMFlag==1),:));
        if any(isnan(CM(cntr,:)))
            AllPtsOnSlice(find(PtsForCMFlag==1),:)
        end
    end
end
close(CoMWaitbarH);
% Plot the center of mass
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
if PlotThings == 1
    hold on
    plot3(CM(:,1),CM(:,2),CM(:,3),'*k')
end
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
CMBackTransfo = (TrafoMtx\CM');
```

ControlFlags.m

```
function ControlFlags(GlobalFaceFlag,GlobalLineFlag,PlotFlag)
disp('ControlFlags...');
if PlotFlag==1
    %% Calling Global Values
    %--------------------------------------------------------------------
    global Param;
    global F; global V;
    %% Do another figure "Just for control"
    %--------------------------------------------------------------------
    FigH = figure('Position',Param.FigPos);
    set(FigH,'Name','ControlFlags','NumberTitle','off')
    for i = 1:length(unique(GlobalFaceFlag))
        SurfH = trisurf(F(find(GlobalFaceFlag==i-1),:),...
                        V(:,1),V(:,2),V(:,3),i-1);
        LegEntry{i} = ['Face ',num2str(i-1)];
        hold on
    end
    Colors = {'*r','*m','*b','*k','*c','*y'};
    for j = 1:length(unique(GlobalFaceFlag))-1
plot3(V(find(GlobalLineFlag==j),1),V(find(GlobalLineFlag==j),2),...
             V(find(GlobalLineFlag==j),3),Colors{j})
        LegEntry{i+j} = ['Line ',num2str(j)];
    end
    legend(LegEntry);
    axis equal; axis vis3d;

    if isinf(Param.WaitTimeout)
        uiwait(FigH)
    else
        uiwait(FigH,Param.WaitTimeout)
    end
end
```

CreateTheCenterline.m

```
function CLineFun = CreateTheCenterline(CoM,PlotFlag)
disp('CreateTheCenterline...')

%% Global Values
%--------------------------------------------------------------------------
global Param;
global F; global V;
global SurfH
global Name

%% Create the centerline
%--------------------------------------------------------------------------
CLineFun = cscvn(CoM');
save([Name.StlFile_Name,'_3_CLineFun'],'CLineFun');

%% Do the interpolation
%--------------------------------------------------------------------------
Interp_t  = linspace(CLineFun.breaks(1),CLineFun.breaks(end),...
                  10*size(CLineFun.breaks,2));
Interp_pt = ppval(CLineFun,Interp_t);

%% Create the figure for the vessel and its CoM
%--------------------------------------------------------------------------
if PlotFlag == 1
    H.ClineFig = figure('Position',Param.FigPos,...
        'NumberTitle','off',...
        'Name','CreateTheCenterline: Check The Centerline');
    SurfH      = trisurf(F,V(:,1),V(:,2),V(:,3),0,'FaceAlpha',0.1,...
                         'EdgeAlpha',0.1);
    hold on;
    ComH       =
plot3(CoM(:,1),CoM(:,2),CoM(:,3),'ob','MarkerFaceColor','b');
    ClineH     = plot3(Interp_pt(1,:),Interp_pt(2,:),Interp_pt(3,:),'-
b');
    axis equal; axis vis3d;
    drawnow
end
```

Do1DCoil.m

```
function Do1DCoil(RadFun,CLineFun,PlotFlag)
disp('Do1DCoil...')

%% Global Values
%--------------------------------------------------------------------------
global FinalCAS;
global Stent;
global Param;
global CoilPts
global Name

%% See if the variables exists
%--------------------------------------------------------------------------
if exist([Name.StlFile_Name,'_8_CoilPts.mat'],'file')
    Temp = load([Name.StlFile_Name,'_8_CoilPts.mat'],'CoilPts');
    if isfield(Temp.CoilPts,'Coord')
        CoilPts = Temp.CoilPts;
        Plot(CoilPts,PlotFlag)
        disp(['=> "CoilPts" from ',Name.StlFile_Name,...
            '_8_CoilPts.mat have been loaded'])
        return
    end
end

%% Initialisation
%--------------------------------------------------------------------------
CASNrOf   = floor(Stent.NOfTurns*2*Stent.CoilNrOf*12+1);
Alpha     = (2*pi/(Stent.CoilNrOf*12)); % Angle Between Cstrct planes
Betas     = (0:CASNrOf-1)*Alpha/2;              % Angle change Between
Cstrct planes
GammaL    = (0.5+(0:Stent.CoilNrOf-1))*(2*pi/(Stent.CoilNrOf)); % Offset
at t=0
GammaR    = (0:Stent.CoilNrOf-1)*(2*pi/(Stent.CoilNrOf)); % Offset at t=0
Rad       = ppval(RadFun,FinalCAS.CumulDist(1:CASNrOf));
LofCL     =
LengthOfcscvn(CLineFun,CLineFun.breaks(1),CLineFun.breaks(end),...
                Param.LengthOfcscvnTol);
if FinalCAS.CumulDist(CASNrOf)>LofCL
    RplcmtIndx = find(FinalCAS.CumulDist(1:CASNrOf)>LofCL);
    Rad(RplcmtIndx) = Rad(RplcmtIndx(1));
end

% Make the TrafoSuperMtx
TrafoSuperMtx = zeros(3*CASNrOf,3*CASNrOf);
for i = 1:3:(CASNrOf)*3
    TrafoSuperMtx(i:i+2,i)   = FinalCAS.E1(1:3,ceil(i/3));
    TrafoSuperMtx(i:i+2,i+1) = FinalCAS.E2(1:3,ceil(i/3));
    TrafoSuperMtx(i:i+2,i+2) = FinalCAS.E3(1:3,ceil(i/3));
end
OrigSuperMtx = reshape(FinalCAS.Origin(:,1:CASNrOf),1,[])';

%% Produce the coils
%--------------------------------------------------------------------------
for i=1:Stent.CoilNrOf
```

```
    % RightPTs
    PtsR = [zeros(size(Rad));...
            Rad.*cos(Alpha+Betas+GammaR(i));...
            Rad.*sin(Alpha+Betas+GammaR(i))];
    % LeftPTs
    PtsL = [zeros(size(Rad));...
            Rad.*cos(Alpha-Betas+GammaL(i));...
            Rad.*sin(Alpha-Betas+GammaL(i))];
    ActPtsL =
reshape(TrafoSuperMtx*reshape(PtsL,1,[])'+OrigSuperMtx,3,[]);
    ActPtsR =
reshape(TrafoSuperMtx*reshape(PtsR,1,[])'+OrigSuperMtx,3,[]);
    CoilPts(i,1).Coord = ActPtsR;
    CoilPts(i,2).Coord = ActPtsL;
end
save([Name.StlFile_Name,'_8_CoilPts.mat'],'CoilPts');
Plot(CoilPts,PlotFlag)
end
function Plot(CoilPts,PlotFlag)
global Param
global Stent
global F; global V;
%% Plot
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
if PlotFlag == 1
    figure('Position',Param.FigPos,'Name','Do1DCoil',...
        'NumberTitle','off');
    trisurf(F,V(:,1),V(:,2),V(:,3),0,'FaceAlpha',0.0,'EdgeAlpha',0.1);
    hold on;
    for i=1:Stent.CoilNrOf
plot3(CoilPts(i,1).Coord(1,:),CoilPts(i,1).Coord(2,:),CoilPts(i,1).Coord(
3,:),'-r')
plot3(CoilPts(i,2).Coord(1,:),CoilPts(i,2).Coord(2,:),CoilPts(i,2).Coord(
3,:),'-b')
        hold on
    end
    axis equal; grid on; axis vis3d
    plotbrowser('on')
end
end
```

DoCoilCircles.m

```
function CPts = DoCoilCircles()
global Param
Angles  = (0:(2*pi/Param.StrutsPtsNrOf):2*pi-(2*pi/Param.StrutsPtsNrOf));
CPts    =
[(Param.dCoil/2)*sin(Angles);(Param.dCoil/2)*cos(Angles);zeros(1,Param.St
rutsPtsNrOf)];
```

DoCoilEllipses.m

```
function [EPts,ePts] = DoCoilEllipses(v1,v2)
global Param
alpha  = acos(dot(v1,v2));
Angles = (0:(2*pi/Param.StrutsPtsNrOf):2*pi-(2*pi/Param.StrutsPtsNrOf));
RE     = sin(alpha/2)*Param.dCoil/sin(alpha);
rE     = cos(alpha/2)*Param.dCoil/sin(alpha);
EPts   =
[RE*sin(Angles);(Param.dCoil/2)*cos(Angles);zeros(1,Param.StrutsPtsNrOf)]
;
ePts   =
[rE*sin(Angles);(Param.dCoil/2)*cos(Angles);zeros(1,Param.StrutsPtsNrOf)]
;
```

DoIntraXing.m

```
function CPts = DoIntraXing(CNr,Xid,RL)
global CoilPts
global FinalCAS
global Stent

PlotFlag = 0;
%% Segment from the midline to the Xing => W3
%-----------------------------------------
OCSeg = CoilPts(CNr,RL).Coord(:,Xid) - FinalCAS.Origin(:,Xid);
W3    = OCSeg/norm(OCSeg);
%% W1 & W2
%-----------------------------------------
% Segment from the previous point to the next of this line
if Xid == 1
    % For the Segment at the very beginning
    PNSeg1 = CoilPts(CNr,RL).Coord(:,Xid+1)-CoilPts(CNr,RL).Coord(:,Xid);
else
    % For all other segments
    PNSeg1 = CoilPts(CNr,RL).Coord(:,Xid+1)-CoilPts(CNr,RL).Coord(:,Xid-
1);
end
```


```
v1     = PNSeg1/norm(PNSeg1);
W1 = cross(v1,W3)/norm(cross(v1,W3));
W2 = cross(W3,W1)/norm(cross(W1,W3));
% Prepare Points and the transfomatrix
%-------------------------------------
tt  = linspace(0,2*pi,7);
Pts = [zeros(1,length(tt)-1);
       Stent.CRad*sin(tt(1:end-1));
       Stent.CRad*cos(tt(1:end-1))];
TrafoMtx = [W2,W1,W3];
Orig     = CoilPts(CNr,RL).Coord(:,Xid);
% Transformed points
%--------------------
CPts = TrafoMtx*Pts+repmat(Orig,1,length(tt)-1);
if PlotFlag == 1
    plot3(CPts(1,:),CPts(2,:),CPts(3,:),'.-m')
    hold on
%     quiver3(Orig(1),Orig(2),Orig(3),W1(1),W1(2),W1(3),'r')
%     quiver3(Orig(1),Orig(2),Orig(3),W2(1),W2(2),W2(3),'g')
%     quiver3(Orig(1),Orig(2),Orig(3),W3(1),W3(2),W3(3),'b')
    axis equal; grid on; box on;
end
end
```

DoMesh.m

```
function DoMesh()
disp('DoMesh...')

global CoilPts
global MeshVertices
global Param
PlotLines = 0;
PlotXing  = 0;
PlotIntra = 0;
PlotFaces = 0;
WriteSTLFlag = 1;
% global MaxCoils
MaxCoils  = 24;
% MaxPts    = 25*12+1;
MaxPts   = size(CoilPts(1,1).Coord(1,:),2)-3;
MaxElmtLen = 0.2;
%% Plot
%--------------------------------------------------------------------------
if PlotLines == 1
    figure('Position',Param.FigPos);
    % for i=1:Stent.CoilNrOf
    for i=1:MaxCoils
        plot3(CoilPts(i,1).Coord(1,1:MaxPts),...
              CoilPts(i,1).Coord(2,1:MaxPts),...
              CoilPts(i,1).Coord(3,1:MaxPts),'-r');
        hold on
        plot3(CoilPts(i,2).Coord(1,1:MaxPts),...
              CoilPts(i,2).Coord(2,1:MaxPts),...
              CoilPts(i,2).Coord(3,1:MaxPts),'-b');
    end
end


% Crossing Ids
XIds = 7:12:MaxPts;
% Basic indices for the Faces
BaseIndx = [ 1, 2, 7; 2, 3, 8; 3, 4, 9; 4, 5,10; 5, 6,11; 6, 1,12;...
             7, 8, 2; 8, 9, 3; 9,10, 4;10,11, 5;11,12, 6;12, 7, 1];
for CoilNr=1:MaxCoils;
    % Plot the Crossing
    %------------------
    if PlotXing == 1
plot3(CoilPts(CoilNr,1).Coord(1,7:12:MaxPts),CoilPts(CoilNr,1).Coord(2,7:
12:MaxPts),CoilPts(CoilNr,1).Coord(3,7:12:MaxPts),'*k');
        hold on
plot3(CoilPts(CoilNr,2).Coord(1,7:12:MaxPts),CoilPts(CoilNr,2).Coord(2,7:
12:MaxPts),CoilPts(CoilNr,2).Coord(3,7:12:MaxPts),'*k');
    end
    % Initialisation
    %--------------------------------------------------------------------------
    % Vertice Matrix
    MeshVertices(CoilNr,1).Coord = [];
    MeshVertices(CoilNr,2).Coord = [];
    % Face Matrix
```

```
    MeshFaces(CoilNr,1).Id = [];
    MeshFaces(CoilNr,2).Id = [];
    % Distances Between the points
    DistR = sqrt(sum((CoilPts(CoilNr,1).Coord(:,2:end)-...
                     CoilPts(CoilNr,1).Coord(:,1:end-1)).^2,1));
    DistL = sqrt(sum((CoilPts(CoilNr,2).Coord(:,2:end)-...
                     CoilPts(CoilNr,2).Coord(:,1:end-1)).^2,1));
    %% Do the Segments before the first Xing
    %-----------------------------------------------------------------
    % Distance to next Xing
    LBtwXR   = sum(DistR(1:XIds(1)-1));
    LBtwXL   = sum(DistL(1:XIds(1)-1));
    % Number of segments to next Xing
    SegNrOfR = min(ceil(LBtwXR/MaxElmtLen),2);
    SegNrOfL = min(ceil(LBtwXL/MaxElmtLen),2);
%     SegNrOfR = 2;
%     SegNrOfL = 2;
    switch SegNrOfR
        case 2
%*************************************************************
            % Get the Ids where a segment has to be made
            IntraId = 1:3:XIds(1);
            IntraId = IntraId(1:end-1);
            if PlotIntra == 1
                plot3(CoilPts(CoilNr,1).Coord(1,IntraId),...
                      CoilPts(CoilNr,1).Coord(2,IntraId),...
                      CoilPts(CoilNr,1).Coord(3,IntraId),'.m');
                hold on;
            end
            % Do and register the first vertices
            RightPts = DoIntraXing(CoilNr,IntraId(1),1);
            MeshVertices(CoilNr,1).Coord(:,end+1:end+6) = RightPts;
            % Do and register the next vertices
            for i=2:2
                RightPts = DoIntraXing(CoilNr,IntraId(i),1);
                MeshVertices(CoilNr,1).Coord(:,end+1:end+6) = RightPts;
                % Do and register the Face Indices
                NSegments = size(MeshVertices(CoilNr,1).Coord,2)/6;
                NewIndx = BaseIndx+(NSegments-2)*6*ones(size(BaseIndx));
                MeshFaces(CoilNr,1).Id(end+1:end+12,:) = NewIndx;
            end
%*************************************************************
        case 1
%*************************************************************
            % Get the Ids where a segment has to be made
            IntraId = 1:6:XIds(1);
            IntraId = IntraId(1:end-1);
            if PlotIntra == 1
                text(CoilPts(CoilNr,1).Coord(1,4),...
                     CoilPts(CoilNr,1).Coord(2,4),...
                     CoilPts(CoilNr,1).Coord(3,4),'1')
                hold on;
            end
            % Do and register the first vertices
            RightPts = DoIntraXing(CoilNr,IntraId(1),1);
            MeshVertices(CoilNr,1).Coord(:,end+1:end+6) = RightPts;
%*************************************************************
    end
```

```
    switch SegNrOfL
        case 2
%*************************************************************
            % Get the Ids where a segment has to be made
            IntraId = 1:3:XIds(1);
            IntraId = IntraId(1:end-1);
            if PlotIntra == 1
                plot3(CoilPts(CoilNr,2).Coord(1,IntraId),...
                      CoilPts(CoilNr,2).Coord(2,IntraId),...
                      CoilPts(CoilNr,2).Coord(3,IntraId),'.m');
                hold on;
            end
            % Do and register the first vertices
            LeftPts = DoIntraXing(CoilNr,IntraId(1),2);
            MeshVertices(CoilNr,2).Coord(:,end+1:end+6) = LeftPts;
            % Do and register the next vertices
            for i=2:2
                LeftPts = DoIntraXing(CoilNr,IntraId(i),2);
                MeshVertices(CoilNr,2).Coord(:,end+1:end+6) = LeftPts;
                % Do and register the Face Indices
                NSegments = size(MeshVertices(CoilNr,2).Coord,2)/6;
                NewIndx = BaseIndx+(NSegments-2)*6*ones(size(BaseIndx));
                MeshFaces(CoilNr,2).Id(end+1:end+12,:) = NewIndx;
            end
%*************************************************************
        case 1
%*************************************************************
            % Get the Ids where a segment has to be made
            IntraId = 1:6:XIds(1);
            IntraId = IntraId(1:end-1);
            if PlotIntra == 1
                text(CoilPts(CoilNr,2).Coord(1,4),...
                     CoilPts(CoilNr,2).Coord(2,4),...
                     CoilPts(CoilNr,2).Coord(3,4),'1')
                hold on;
            end
            % Do and register the first vertices
            LeftPts = DoIntraXing(CoilNr,IntraId(1),2);
            MeshVertices(CoilNr,2).Coord(:,end+1:end+6) = LeftPts;
%*************************************************************
    end
    %% Do the Middle Segments
    %-------------------------------------------------------------------
    for XingNr = 1:size(XIds,2)-1
        % Crossing
        %-------------------------------------------------------------------
        % Do and register the Crossing vertices
        [RightPts1,RightPts2] = DoXIng(CoilNr,XIds(XingNr),XingNr,1);
        [LeftPts1,LeftPts2]   = DoXIng(CoilNr,XIds(XingNr),XingNr,2);
%         LeftPts1 = ones(3,6);
%         LeftPts2 = ones(3,6);
        MeshVertices(CoilNr,1).Coord(:,end+1:end+6) = RightPts1;
        MeshVertices(CoilNr,2).Coord(:,end+1:end+6) = LeftPts1;
        % Do and register the Face Indices
        NSegments = size(MeshVertices(CoilNr,1).Coord,2)/6;
        NewIndx = BaseIndx+(NSegments-2)*6*ones(size(BaseIndx));
        MeshFaces(CoilNr,1).Id(end+1:end+12,:) = NewIndx;
        NSegments = size(MeshVertices(CoilNr,2).Coord,2)/6;
```

```
        NewIndx = BaseIndx+(NSegments-2)*6*ones(size(BaseIndx));
        MeshFaces(CoilNr,2).Id(end+1:end+12,:) = NewIndx;
        MeshVertices(CoilNr,1).Coord(:,end+1:end+6) = RightPts2;
        MeshVertices(CoilNr,2).Coord(:,end+1:end+6) = LeftPts2;
        % Between Crossing
        %-----------------------------------------------------------------
        % Distance to next Xing
        LBtwXR   = sum(DistR(XIds(XingNr):XIds(XingNr+1)-1));
        LBtwXL   = sum(DistL(XIds(XingNr):XIds(XingNr+1)-1));
        % Number of segments to next Xing
        SegNrOfR = min(ceil(LBtwXR/MaxElmtLen),4);
        SegNrOfL = min(ceil(LBtwXL/MaxElmtLen),4);
%         SegNrOfR = 2;
%         SegNrOfL = 4;
        switch SegNrOfR
            case 4
%*******************************************************
                % Get the Ids where a segment has to be made
                IntraIdR = XIds(XingNr):3:XIds(XingNr+1);
                IntraIdR = IntraIdR(2:end-1);
                if PlotIntra == 1
                    plot3(CoilPts(CoilNr,1).Coord(1,IntraIdR),...
                          CoilPts(CoilNr,1).Coord(2,IntraIdR),...
                          CoilPts(CoilNr,1).Coord(3,IntraIdR),'.k');
                    hold on;
                end
                for i=1:3
                    % Do and register the vertices
                    RightPts = DoIntraXing(CoilNr,IntraIdR(i),1);
                    MeshVertices(CoilNr,1).Coord(:,end+1:end+6) =
RightPts;
                    % Do and register the Face Indices
                    NSegments = size(MeshVertices(CoilNr,1).Coord,2)/6;
                    NewIndx = BaseIndx+(NSegments-
2)*6*ones(size(BaseIndx));
                    MeshFaces(CoilNr,1).Id(end+1:end+12,:) = NewIndx;
                end
%*******************************************************
            case 3
%*******************************************************
                % Get the Ids where a segment has to be made
                IntraIdR = XIds(XingNr):4:XIds(XingNr+1);
                IntraIdR = IntraIdR(2:end-1);
                if PlotIntra == 1
                    plot3(CoilPts(CoilNr,1).Coord(1,IntraIdR),...
                          CoilPts(CoilNr,1).Coord(2,IntraIdR),...
                          CoilPts(CoilNr,1).Coord(3,IntraIdR),'.k');
                    hold on;
                end
                for i=1:2
                    % Do and register the vertices
                    RightPts = DoIntraXing(CoilNr,IntraIdR(i),1);
                    MeshVertices(CoilNr,1).Coord(:,end+1:end+6) =
RightPts;
                    % Do and register the Face Indices
                    NSegments = size(MeshVertices(CoilNr,1).Coord,2)/6;
                    NewIndx = BaseIndx+(NSegments-
2)*6*ones(size(BaseIndx));
```

```
                    MeshFaces(CoilNr,1).Id(end+1:end+12,:) = NewIndx;
                end
%************************************************************
            case 2
%*****************************************************
                % Get the Ids where a segment has to be made
                IntraIdR = XIds(XingNr):6:XIds(XingNr+1);
                IntraIdR = IntraIdR(2:end-1);
                if PlotIntra == 1
                    plot3(CoilPts(CoilNr,1).Coord(1,IntraIdR),...
                          CoilPts(CoilNr,1).Coord(2,IntraIdR),...
                          CoilPts(CoilNr,1).Coord(3,IntraIdR),'.k');
                    hold on;
                end
                for i=1:1
                    % Do and register the vertices
                    RightPts = DoIntraXing(CoilNr,IntraIdR(i),1);
                    MeshVertices(CoilNr,1).Coord(:,end+1:end+6) =
RightPts;
                    % Do and register the Face Indices
                    NSegments = size(MeshVertices(CoilNr,1).Coord,2)/6;
                    NewIndx = BaseIndx+(NSegments-
2)*6*ones(size(BaseIndx));
                    MeshFaces(CoilNr,1).Id(end+1:end+12,:) = NewIndx;
                end
%************************************************************
            case 1
%*****************************************************
                if XingNr == size(XIds,2)-1
                    % Get the Ids where a segment has to be made
                    IntraIdR = XIds(XingNr):6:XIds(XingNr+1);
                    IntraIdR = IntraIdR(2:end-1);
                    % Do and register the vertices
                    RightPts = DoIntraXing(CoilNr,IntraIdR(1),1);
                    MeshVertices(CoilNr,1).Coord(:,end+1:end+6) =
RightPts;
                    % Do and register the Face Indices
                    NSegments = size(MeshVertices(CoilNr,1).Coord,2)/6;
                    NewIndx = BaseIndx+(NSegments-
2)*6*ones(size(BaseIndx));
                    MeshFaces(CoilNr,1).Id(end+1:end+12,:) = NewIndx;
                end
                if PlotIntra == 1
                    text(CoilPts(CoilNr,1).Coord(1,XIds(XingNr)+6),...
                         CoilPts(CoilNr,1).Coord(2,XIds(XingNr)+6),...
                         CoilPts(CoilNr,1).Coord(3,XIds(XingNr)+6),'1')
                    hold on;
                end
%************************************************************
        end
        switch SegNrOfL
            case 4
%*****************************************************
                % Get the Ids where a segment has to be made
                IntraIdL = XIds(XingNr):3:XIds(XingNr+1);
                IntraIdL = IntraIdL(2:end-1);
                if PlotIntra == 1
                    plot3(CoilPts(CoilNr,2).Coord(1,IntraIdL),...
```

```
                        CoilPts(CoilNr,2).Coord(2,IntraIdL),...
                        CoilPts(CoilNr,2).Coord(3,IntraIdL),'.k');
                    hold on;
                end
                for i=1:3
                    % Do and register the vertices
                    LeftPts = DoIntraXing(CoilNr,IntraIdL(i),2);
                    MeshVertices(CoilNr,2).Coord(:,end+1:end+6)=LeftPts;
                    % Do and register the Face Indices
                    NSegments = size(MeshVertices(CoilNr,2).Coord,2)/6;
                    NewIndx = BaseIndx+(NSegments-
2)*6*ones(size(BaseIndx));
                    MeshFaces(CoilNr,2).Id(end+1:end+12,:) = NewIndx;
                end
%************************************************************
            case 3
%******************************************************
                % Get the Ids where a segment has to be made
                IntraIdL = XIds(XingNr):4:XIds(XingNr+1);
                IntraIdL = IntraIdL(2:end-1);
                if PlotIntra == 1
                    plot3(CoilPts(CoilNr,2).Coord(1,IntraIdL),...
                        CoilPts(CoilNr,2).Coord(2,IntraIdL),...
                        CoilPts(CoilNr,2).Coord(3,IntraIdL),'.k');
                    hold on;
                end
                for i=1:2
                    % Do and register the vertices
                    LeftPts = DoIntraXing(CoilNr,IntraIdL(i),2);
                    MeshVertices(CoilNr,2).Coord(:,end+1:end+6)=LeftPts;
                    % Do and register the Face Indices
                    NSegments = size(MeshVertices(CoilNr,2).Coord,2)/6;
                    NewIndx = BaseIndx+(NSegments-
2)*6*ones(size(BaseIndx));
                    MeshFaces(CoilNr,2).Id(end+1:end+12,:) = NewIndx;
                end
%************************************************************
            case 2
%******************************************************
                % Get the Ids where a segment has to be made
                IntraIdL = XIds(XingNr):6:XIds(XingNr+1);
                IntraIdL = IntraIdL(2:end-1);
                if PlotIntra == 1
                    plot3(CoilPts(CoilNr,2).Coord(1,IntraIdL),...
                        CoilPts(CoilNr,2).Coord(2,IntraIdL),...
                        CoilPts(CoilNr,2).Coord(3,IntraIdL),'.k');
                    hold on;
                end
                for i=1:1
                    % Do and register the vertices
                    LeftPts = DoIntraXing(CoilNr,IntraIdL(i),2);
                    MeshVertices(CoilNr,2).Coord(:,end+1:end+6)=LeftPts;
                    % Do and register the Face Indices
                    NSegments = size(MeshVertices(CoilNr,2).Coord,2)/6;
                    NewIndx = BaseIndx+(NSegments-
2)*6*ones(size(BaseIndx));
                    MeshFaces(CoilNr,2).Id(end+1:end+12,:) = NewIndx;
                end
```

```
%*************************************************************
            case 1
%*************************************************
                if XingNr == size(XIds,2)-1
                    % Get the Ids where a segment has to be made
                    IntraIdL = XIds(XingNr):6:XIds(XingNr+1);
                    IntraIdL = IntraIdL(2:end-1);
                    % Do and register the vertices
                    LeftPts = DoIntraXing(CoilNr,IntraIdR(1),2);
                    MeshVertices(CoilNr,2).Coord(:,end+1:end+6)=LeftPts;
                    % Do and register the Face Indices
                    NSegments = size(MeshVertices(CoilNr,2).Coord,2)/6;
                    NewIndx = BaseIndx+(NSegments-
2)*6*ones(size(BaseIndx));
                    MeshFaces(CoilNr,2).Id(end+1:end+12,:) = NewIndx;
                end
                % Plot the midpoints
                if PlotIntra == 1
                    text(CoilPts(CoilNr,2).Coord(1,XIds(XingNr)+6),...
                         CoilPts(CoilNr,2).Coord(2,XIds(XingNr)+6),...
                         CoilPts(CoilNr,2).Coord(3,XIds(XingNr)+6),'1')
                    hold on;
                end
%*************************************************************
        end
    end
    %% Close the Coils ends
    %-----------------------------------------------------------------------
    % Right End
    CenterEndPtR = CoilPts(CoilNr,1).Coord(:,IntraIdR(end));
    MeshVertices(CoilNr,1).Coord(:,end+1) = CenterEndPtR;
    CenterEndIdR = size(MeshVertices(CoilNr,1).Coord,2);
    NewIndx = [CenterEndIdR-6,CenterEndIdR-5,CenterEndIdR;...
               CenterEndIdR-5,CenterEndIdR-4,CenterEndIdR;...
               CenterEndIdR-4,CenterEndIdR-3,CenterEndIdR;...
               CenterEndIdR-3,CenterEndIdR-2,CenterEndIdR;...
               CenterEndIdR-2,CenterEndIdR-1,CenterEndIdR;...
               CenterEndIdR-1,CenterEndIdR-6,CenterEndIdR];
    MeshFaces(CoilNr,1).Id(end+1:end+6,:) = NewIndx;
    % Right Beginning
    CenterBeginPtR = CoilPts(CoilNr,1).Coord(:,1);
    MeshVertices(CoilNr,1).Coord(:,end+1) = CenterBeginPtR;
    CenterBeginIdR = size(MeshVertices(CoilNr,1).Coord,2);
    NewIndx =
[1,2,CenterBeginIdR;2,3,CenterBeginIdR;3,4,CenterBeginIdR;...
               4,5,CenterBeginIdR;5,6,CenterBeginIdR;6,1,CenterBeginIdR];
    MeshFaces(CoilNr,1).Id(end+1:end+6,:) = NewIndx;
    % Left End
    CenterEndPtL = CoilPts(CoilNr,2).Coord(:,IntraIdL(end));
    MeshVertices(CoilNr,2).Coord(:,end+1) = CenterEndPtL;
    CenterEndIdL = size(MeshVertices(CoilNr,2).Coord,2);
    NewIndx = [CenterEndIdL-6,CenterEndIdL-5,CenterEndIdL;...
               CenterEndIdL-5,CenterEndIdL-4,CenterEndIdL;...
               CenterEndIdL-4,CenterEndIdL-3,CenterEndIdL;...
               CenterEndIdL-3,CenterEndIdL-2,CenterEndIdL;...
               CenterEndIdL-2,CenterEndIdL-1,CenterEndIdL;...
               CenterEndIdL-1,CenterEndIdL-6,CenterEndIdL];
    MeshFaces(CoilNr,2).Id(end+1:end+6,:) = NewIndx;
```

```
    % Left Beginning
    CenterBeginPtL = CoilPts(CoilNr,2).Coord(:,1);
    MeshVertices(CoilNr,2).Coord(:,end+1) = CenterBeginPtL;
    CenterBeginIdL = size(MeshVertices(CoilNr,2).Coord,2);
    NewIndx =
[1,2,CenterBeginIdL;2,3,CenterBeginIdL;3,4,CenterBeginIdL;...
              4,5,CenterBeginIdL;5,6,CenterBeginIdL;6,1,CenterBeginIdL];
    MeshFaces(CoilNr,2).Id(end+1:end+6,:) = NewIndx;
end
if or(PlotFaces,WriteSTLFlag) == 1;
    for CoilNr=1:MaxCoils;
        VR = MeshVertices(CoilNr,1).Coord';
        FR = MeshFaces(CoilNr,1).Id;
        VL = MeshVertices(CoilNr,2).Coord';
        FL = MeshFaces(CoilNr,2).Id;
        if PlotFaces == 1
            trisurf(FR,VR(:,1),VR(:,2),VR(:,3),2*ones(1,size(FR,1)))
            hold on
            trisurf(FL,VL(:,1),VL(:,2),VL(:,3),1*ones(1,size(FL,1)))
            hold on
            axis equal; grid on; axis vis3d
            drawnow
        end
        if WriteSTLFlag==1
            if CoilNr == 1 % the file has to be opened
                DoStlFile('test',FR,VR,1)
                DoStlFile('test',FL,VL,0)
            elseif CoilNr == MaxCoils; % the file has to be closed
                DoStlFile('test',FR,VR,0)
                DoStlFile('test',FL,VL,2)
            else
                DoStlFile('test',FR,VR,0)
                DoStlFile('test',FL,VL,0)
            end
        end
    end
end
end
```

DoNoXingSegments.m

```
function [MeshPts,MeshCells,LMeshPts,RMeshPts] =...
    DoNoXingSegments(MeshPts,MeshCells,LCNr,i,LCoilFun,RCoilFun,...
    MidLinePts,LMeshPts,RMeshPts)
%(LCNr,i)
global Param
% Corresponding RNr
RCNr = LCNr;
% Number of existing mesh points
CntrLb = size(MeshPts,1);
% Functions
LFun = LCoilFun{LCNr};
LDer = fnder(LFun,1);
RFun = RCoilFun{RCNr};
RDer = fnder(RFun,1);
% Mid line Points
MidPta = MidLinePts(i,:);
MidPtb = MidLinePts(i-1,:);
% Circle pts
CPts = DoCoilCircles();
% Mesh points @ Lbefore
if isnan(LMeshPts(LCNr,i-1,1))
    LPtb   = ppval(LFun,LFun.breaks(i-1))';
    Lbw3   = (LPtb-MidPtb)' / norm(LPtb-MidPtb);
    Lbw1   = ppval(LDer,LFun.breaks(i-1))/norm(ppval(LDer,LFun.breaks(i-
1)));
    Lbw2   = cross(Lbw3,Lbw1)/norm(cross(Lbw3,Lbw1));
    CPts = DoCoilCircles();
    MPtsLb = repmat(LPtb',1,Param.StrutsPtsNrOf) + [Lbw3,Lbw2,Lbw1]*CPts;
    MeshPts(CntrLb+ 1:CntrLb+ 6,1:3) = MPtsLb';
    LMeshPts(LCNr,i-1,1:6) = CntrLb+ 1:CntrLb+ 6;
    CntrLa = CntrLb + 6;
else
    PtNr(1:6) = LMeshPts(LCNr,i-1,:);
    MPtsLb    = MeshPts(PtNr,:)';
    CntrLa    = CntrLb;
end
% Mesh points @ Lact
if isnan(LMeshPts(LCNr,i,1))
    LPta   = ppval(LFun,LFun.breaks(i))';
    Lnw3   = (LPta-MidPta)' / norm(LPta-MidPta);
    Lnw1   = ppval(LDer,LFun.breaks(i))/norm(ppval(LDer,LFun.breaks(i)));
    Lnw2   = cross(Lnw3,Lnw1)/norm(cross(Lnw3,Lnw1));
    MPtsLa = repmat(LPta',1,Param.StrutsPtsNrOf) + [Lnw3,Lnw2,Lnw1]*CPts;
    MeshPts(CntrLa+ 1:CntrLa+ 6,1:3) = MPtsLa';
    LMeshPts(LCNr,i,1:6) = CntrLa+ 1:CntrLa+ 6;
    CntrRb = CntrLa+6;
else
    PtNr(1:6) = LMeshPts(LCNr,i,:);
    MPtsLa    = MeshPts(PtNr,:)';
    CntrRb    = CntrLa;
end
% Mesh points @ Rbefore
if isnan(RMeshPts(RCNr,i-1,1))
    RPtb   = ppval(RFun,RFun.breaks(i-1))';
    Rbw3   = (RPtb-MidPtb)' / norm(RPtb-MidPtb);
```


```
    Rbw1   = ppval(RDer,RFun.breaks(i-1))/norm(ppval(RDer,RFun.breaks(i-
1)));
    Rbw2   = cross(Rbw3,Rbw1)/norm(cross(Rbw3,Rbw1));
    MPtsRb = repmat(RPtb',1,Param.StrutsPtsNrOf) + [Rbw3,Rbw2,Rbw1]*CPts;
    MeshPts(CntrRb+ 1:CntrRb+6,1:3) = MPtsRb';
    RMeshPts(RCNr,i-1,1:6) = CntrRb+ 1:CntrRb+6;
    CntrRa = CntrRb+6;
else
    PtNr(1:6) = RMeshPts(RCNr,i-1,:);
    MPtsRb    = MeshPts(PtNr,:)';
    CntrRa = CntrRb;
end
% Mesh points @ Ract
if isnan(RMeshPts(RCNr,i,1))
    RPta   = ppval(RFun,RFun.breaks(i))';
    Rnw3   = (RPta-MidPta)' / norm(RPta-MidPta);
    Rnw1   = ppval(RDer,RFun.breaks(i))/norm(ppval(RDer,RFun.breaks(i)));
    Rnw2   = cross(Rnw3,Rnw1)/norm(cross(Rnw3,Rnw1));
    MPtsRa = repmat(RPta',1,Param.StrutsPtsNrOf) + [Rnw3,Rnw2,Rnw1]*CPts;
    MeshPts(CntrRa+ 1:CntrRa+ 6,1:3) = MPtsRa';
    RMeshPts(RCNr,i,1:6) = CntrRa+ 1:CntrRa+ 6;
else
    PtNr(1:6) = RMeshPts(RCNr,i,:);
    MPtsRa    = MeshPts(PtNr,:)';
end
% Record Elements
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-1,1),LMeshPts(LCNr,i-
1,2),LMeshPts(LCNr,i,1)];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-1,2),LMeshPts(LCNr,i-
1,3),LMeshPts(LCNr,i,2)];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-1,3),LMeshPts(LCNr,i-
1,4),LMeshPts(LCNr,i,3)];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-1,4),LMeshPts(LCNr,i-
1,5),LMeshPts(LCNr,i,4)];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-1,5),LMeshPts(LCNr,i-
1,6),LMeshPts(LCNr,i,5)];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-1,6),LMeshPts(LCNr,i-
1,1),LMeshPts(LCNr,i,6)];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-
1,2),LMeshPts(LCNr,i,2),LMeshPts(LCNr,i,1)];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-
1,3),LMeshPts(LCNr,i,3),LMeshPts(LCNr,i,2)];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-
1,4),LMeshPts(LCNr,i,4),LMeshPts(LCNr,i,3)];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-
1,5),LMeshPts(LCNr,i,5),LMeshPts(LCNr,i,4)];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-
1,6),LMeshPts(LCNr,i,6),LMeshPts(LCNr,i,5)];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-
1,1),LMeshPts(LCNr,i,1),LMeshPts(LCNr,i,6)];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i-1,1),RMeshPts(RCNr,i-
1,2),RMeshPts(RCNr,i,1)];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i-1,2),RMeshPts(RCNr,i-
1,3),RMeshPts(RCNr,i,2)];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i-1,3),RMeshPts(RCNr,i-
1,4),RMeshPts(RCNr,i,3)];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i-1,4),RMeshPts(RCNr,i-
1,5),RMeshPts(RCNr,i,4)];
```

```
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i-1,5),RMeshPts(RCNr,i-
1,6),RMeshPts(RCNr,i,5)];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i-1,6),RMeshPts(RCNr,i-
1,1),RMeshPts(RCNr,i,6)];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i-
1,2),RMeshPts(RCNr,i,2),RMeshPts(RCNr,i,1)];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i-
1,3),RMeshPts(RCNr,i,3),RMeshPts(RCNr,i,2)];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i-
1,4),RMeshPts(RCNr,i,4),RMeshPts(RCNr,i,3)];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i-
1,5),RMeshPts(RCNr,i,5),RMeshPts(RCNr,i,4)];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i-
1,6),RMeshPts(RCNr,i,6),RMeshPts(RCNr,i,5)];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i-
1,1),RMeshPts(RCNr,i,1),RMeshPts(RCNr,i,6)];
```

DoStlFile.m

```
function DoStlFile(StlFileName,MeshCells,MeshPts,FileFlag)
% FileFlag = 1    =>    A new file is created
% FileFlag = 2    =>    The file has to be closed
if FileFlag == 1
    % Open File
    %----------
    fid = fopen([StlFileName,'.stl'],'w');
    % Write Header line
    %------------------
    fprintf(fid,['solid ',StlFileName,'\n']);
else
    fid = fopen([StlFileName,'.stl'],'a');
end
WaitbarH = waitbar(0,'Produce Stl-file');
for i=1:size(MeshCells,1)
     waitbar(i/size(MeshCells,1),WaitbarH);
    % Model
    % -----
    % facet normal 0.000000e+00 0.000000e+00 -1.000000e+00
    %   outer loop
    %     vertex 1.188092e+02 8.682214e+01 -1.000000e+01
    %     vertex -8.029416e+01 -9.922531e+01 -1.000000e+01
    %     vertex -8.029416e+01 8.682214e+01 -1.000000e+01
    %   endloop
    % endfacet
    PtIDs = MeshCells(i,:);
    Pt1   = MeshPts(PtIDs(1),:);
    Pt2   = MeshPts(PtIDs(2),:);
    Pt3   = MeshPts(PtIDs(3),:);
    FacetNorm = cross(Pt2-Pt1,Pt3-Pt1)/norm(cross(Pt2-Pt1,Pt3-Pt1));
    fprintf(fid,['  facet normal %7e %7e %7e \n'],FacetNorm);
    fprintf(fid,['    outer loop \n']);
    fprintf(fid, '      vertex %7e %7e %7e \n',Pt1);
    fprintf(fid, '      vertex %7e %7e %7e \n',Pt2);
    fprintf(fid, '      vertex %7e %7e %7e \n',Pt3);
    fprintf(fid,['    endloop \n']);
    fprintf(fid,['  endfacet \n']);
end
close(WaitbarH);
if FileFlag == 2
    % Write Closing line
    fprintf(fid,['endsolid',StlFileName]);
end
% Close the file
fclose(fid);
```

DoTheCoilPts.m

```
function [LCoilPts,RCoilPts,MidLinePts,LXingRegistry,RXingRegistry] =...
    DoTheCoilPts(MidLineFun,RadFun)
%% Global Values
%---------------
global Param
global TestFlag
%% Function Derivatives
%----------------------
MidLineDer    = fnder(MidLineFun,1);
MidLineDerDer = fnder(MidLineFun,2);
% Arrays to register the Coil Pts
%--------------------------------
LCoilPts = [];
RCoilPts = [];
% First Xing
%-----------
TotalLen = IntOfcscvn(MidLineFun,0,MidLineFun.breaks(end),Param.tol);
% Initialisations for the loop
%-----------------------------
Lstart = Param.tnstart*TotalLen;
Lend   = Param.tnend*TotalLen;
tstart = FindTForGivenLength(MidLineFun,0,MidLineFun.breaks(end),...
                             Lstart,Param.tol);
tend   = FindTForGivenLength(MidLineFun,0,MidLineFun.breaks(end),...
                             Lend,Param.tol);
NBtw  = Param.NBetweenCrossings;
Alpha = (2*pi)/Param.CoilNrOf; % Intercoil angle
Beta  = 0;                     % Time Varying angle
if mod(NBtw,2)==1
    Gamma = 0.50*Alpha;        % Offset right left turning
else
    Gamma = ((NBtw/2)/(NBtw+1))*Alpha;        % Offset right left turning
end
% Waitbar
WaitH = waitbar(0,'DoTheCoilPts...');
tact     = tstart;
PCounter = 0;
XingNr   = 0;
while tact<=tend
    waitbar(tact/tend,WaitH);
    % Actualise Counter
    PCounter = PCounter+1;
    % Xing Managment
    if tact==tstart
        LXingRegistry(PCounter,1:Param.CoilNrOf) =
zeros(1,Param.CoilNrOf);
        RXingRegistry(PCounter,1:Param.CoilNrOf) =
zeros(1,Param.CoilNrOf);
    else
        if or( mod(PCounter-(NBtw+3)/2  ,NBtw+1)==0 && mod(NBtw,2)==1,...
               mod(PCounter-((NBtw/2)+1),NBtw+1)==0 && mod(NBtw,2)==0
)
           % The Crossing appear @x:
           %########################
           % For NBtw odd
```

```
            % ------------
            % PCounter        1 2 3 4 5 6 7 8 9 ...
            % For NBtw = 1    | x | x | x | x | x | x |
            % LCounter          0   2   4   6   8
            %
            % PCounter        1 2 3 4 5 6 7 8 9 ...
            % For NBtw = 3    | | x | | | x | | | x | | | x |
            % LCounter            0       4       8
            %
            % PCounter        1 2 3 4 5 6 7 8 9 ...
            % For NBtw = 5    | | | x | | | | | x | | | | | x |
            % LCounter              0           6          12
            %
            % LCounter = PCounter - (NBtw+3)/2
            %
            % For NBtw even
            % -------------
            % PCounter        1 2 3 4 5 6 7 8 9 ...
            % For NBtw = 2    | x | | x | | x | | x | | x | | x
            % LCounter          0     3     6     9
            %
            % PCounter        1 2 3 4 5 6 7 8 9 ...
            % For NBtw = 4    | | x | | | | x | | | | x | | | | x
            % LCounter            0         5        10
            %
            % PCounter        1 2 3 4 5 6 7 8 9 ...
            % For NBtw = 6    | | | x | | | | | | x | | | | | | x
            % LCounter              0             7            14
            %
            % LCounter = PCounter - ((NBtw/2)+1)
             XingNr       = XingNr+1;
             LStrutNr     = 1:Param.CoilNrOf;
             RStrutNr     = circshift(1:Param.CoilNrOf,[1,-XingNr+1]);
             LXingRegistry(PCounter,LStrutNr) = RStrutNr;
             RXingRegistry(PCounter,RStrutNr) = LStrutNr;
        else
            LXingRegistry(PCounter,1:Param.CoilNrOf) =
zeros(1,Param.CoilNrOf);
            RXingRegistry(PCounter,1:Param.CoilNrOf) =
zeros(1,Param.CoilNrOf);
        end
    end
    % Get the mid point
    %------------------
    CircleMidPt = ppval(MidLineFun,tact);
    %% Calculate the coordinate system Vectors for current point
    %-----------------------------------------------------------------------
    dMLdt    = ppval(MidLineDer,tact);
    ddMLdtdt = ppval(MidLineDerDer,tact);
    % Coordinate system vectors
    %--------------------------
    if tact==tstart
        % (see
http://en.wikipedia.org/wiki/Frenet%E2%80%93Serret_formulas)
        E1 = dMLdt/norm(dMLdt);
        E2 = (ddMLdtdt-dot(ddMLdtdt,E1)*E1)/...
             norm((ddMLdtdt-dot(ddMLdtdt,E1)*E1));
        E3 = cross(E1,E2) / norm(cross(E1,E2));
```

```
        % Since E2 and E3 are not defined for straight ML
        temp=0;
        while any(isnan(E2))
            temp=temp+1;
            E2 = cross([1;1;temp],E1)/norm(cross([1;1;1],E1));
            E3 = cross(E1,E2) / norm(cross(E1,E2));
        end
    else
        % (see Normal orientation methods for 3D offset curves,sweep
        %  surfaces and skinning, Computer Graphics Forum, Volume:
vol.11,
        %  no.3, Pages: C449-57, Published: 1992)
        E1 = dMLdt/norm(dMLdt);
        E2 = E2-dot(E2,E1)*E1;
        E3 = cross(E1,E2) / norm(cross(E1,E2));
    end
    %% Prepare a set of points arranged in circle on the z=0 plane and do
    % their projection on the right node
    %-------------------------------------------------------------------
    Ract    = ppval(RadFun,tact)-Param.RadiusOffset;
    % Left turning coils
    LAngles  = (0:Alpha:2*pi-Alpha)+repmat(Beta,1,Param.CoilNrOf);
    LCircPts =
[Ract*sin(LAngles);Ract*cos(LAngles);zeros(1,Param.CoilNrOf)];
    LProjPts = repmat(CircleMidPt,1,Param.CoilNrOf)+[E3,E2,E1]*LCircPts;
    % Right turning coils
    RAngles  = (0:Alpha:2*pi-Alpha)+repmat(-
Beta,1,Param.CoilNrOf)+repmat(Gamma,1,Param.CoilNrOf);
    RCircPts =
[Ract*sin(RAngles);Ract*cos(RAngles);zeros(1,Param.CoilNrOf)];
    RProjPts = repmat(CircleMidPt,1,Param.CoilNrOf)+[E3,E2,E1]*RCircPts;
    % Save the points
    LCoilPts(:,PCounter,:) = LProjPts';
    RCoilPts(:,PCounter,:) = RProjPts';
    MidLinePts(PCounter,:) = CircleMidPt;
    %% Actualisation of tact
    %-------------------------------------------------------------------
    bact       = sqrt(Param.b0^2+(Param.OrigStentDiam/2)^2-Ract^2);
    LTilNextPt = bact*0.5*Alpha/(1+NBtw);
%     dt         =
FindTForGivenLength(MidLineFun,tact,LTilNextPt,Param.tol)
    BetaOld    = Beta;
    Beta       = BetaOld+0.5*Alpha/(1+NBtw);
    % Case differentiation to be sure to plot the last points of the
stent
    if tact==tend
        % The last set of points that have been plotted is at the end of
        % the Midline => set tact to Infinity so that the loop ends
        tact = Inf;
    else
        % Actualise the next tact
        told = tact;
        tact = FindTForGivenLength(MidLineFun,tact,...
                           MidLineFun.breaks(end),LTilNextPt,Param.tol);
        IsNextAXing = mod(PCounter+1,NBtw+1)==0;
        if IsNextAXing==1
            if tend-tact < 1.5*(tact-told)
                % If there is not enough time to do another crossing
```

```
                % end the procedure here
                tact = Inf;
                disp( 'DoTheCoilPts: Not Enough space for last Xing')
            end
%         else
%             % To avoid having a very short segment at the end of the
coil:
%             % if the remaining part of the line is smaller that half a
%             % "normal" element size, then directly jump to the end
%             if tend-tact < 0.25*(told-tact)
%                 Beta = BetaOld+tend-told;
%                 tact = tend;
%                 disp('DoTheCoilPts: Last short segment avoided')
%             elseif tend-tact < 1*(told-tact) && tend-tact > 0.75*(told-
tact)
%                 tact = Inf;
%                 disp('DoTheCoilPts: Last long segment avoided')
%             end
        end
    end
end
close(WaitH) ;
```

DoTheStentMesh.m

```
function [MeshCells,MeshPts] =
DoTheStentMesh(LCoilPts,RCoilPts,MidLinePts,LCoilFun,RCoilFun,LXingRegist
ry)
global Param
global Plot
LMeshPts  = NaN*ones(size(LCoilPts,1),size(LCoilPts,2),6);
RMeshPts  = NaN*ones(size(RCoilPts,1),size(RCoilPts,2),6);
MeshPts   = [];
MeshCells = [];
% Waitbar
WaitDoStentMeshH = waitbar(0,'Do the Stent Mesh...');
for LCNr = 1:Param.CoilNrOf % Left Coil Nr
     waitbar(LCNr/Param.CoilNrOf,WaitDoStentMeshH);
    if LXingRegistry(end,1)==0
        iMax = size(MidLinePts,1);
    else
        iMax = size(MidLinePts,1)-1;
    end
    for i=1:iMax
        if i==1
            % First Segment => No Crossing
            %-----------------------------
            % Close the coild for L and corresponding R
            [MeshPts,MeshCells] =
CloseCapsBegin(MeshPts,MeshCells,LCNr,i,LCoilFun,RCoilFun,MidLinePts);
        elseif LXingRegistry(i,LCNr)==0 && LXingRegistry(i-1,LCNr)==0
            % No Crossing
            %------------
            [MeshPts,MeshCells,LMeshPts,RMeshPts] =
DoNoXingSegments(MeshPts,MeshCells,...
                LCNr,i,LCoilFun,RCoilFun,...
                MidLinePts,LMeshPts,RMeshPts);
        elseif LXingRegistry(i,LCNr)~=0
            % Crossing should take place (unless its the last segment)
            %--------------------------------------------------------
            if i~=iMax
                % It is not the last segment => Crossing
                %---------------------------------------
                [MeshPts,MeshCells,LMeshPts,RMeshPts] =...
                    DoXingSegments(MeshPts,MeshCells,LCNr,i,LCoilFun,...
                    RCoilFun,MidLinePts,LMeshPts,RMeshPts,LXingRegistry);
            else
                % It is the last segment => No Crossing
                %--------------------------------------
                [MeshPts,MeshCells,LMeshPts,RMeshPts] =
DoNoXingSegments(MeshPts,MeshCells,...
                    LCNr,i,LCoilFun,RCoilFun,...
                    MidLinePts,LMeshPts,RMeshPts);
            end
        end
    end
end
close(WaitDoStentMeshH);
for LCNr = 1:Param.CoilNrOf % Left Coil Nr
```

```
    i=iMax;
    % Close Strut at the end
    %-----------------------
    [MeshPts,MeshCells] = CloseCapsEnd(MeshPts,MeshCells,LCNr,...
        i,LCoilFun,RCoilFun,LMeshPts,RMeshPts);
end
% Plot the mesh Cells
%--------------------
if Plot.Cells ==1
    trisurf(MeshCells,MeshPts(:,1),MeshPts(:,2),MeshPts(:,3),0.1);
    axis equal
end
% Show the Vertice Nr
%--------------------
if Plot.VertNr == 1
    for i=1:size(MeshPts,1)
        text(MeshPts(i,1),MeshPts(i,2),MeshPts(i,3),num2str(i))
    end
end
```

DoXing.m

```
function [EntryPts,ExitPts] = DoXIng(CNr,Xid,XingNr,RL)
global Stent
global FinalCAS
global CoilPts
global MaxCoils
PlotVectors = 0;
Fac = 0.5;
if RL ==1
    LR = 2;
    CrossedId = circshift(1:Stent.CoilNrOf,[0,2-CNr-XingNr]);
%     XingNr = back number of Stockwerk
%     CNr    = back number of name fo coil
%     Fac = 0.1;
%     plot3(CoilPts(CrossedId(1),LR).Coord(1,Xid+1),...
%           CoilPts(CrossedId(1),LR).Coord(2,Xid+1),...
%           CoilPts(CrossedId(1),LR).Coord(3,Xid+1),'ob')
%     hold on
%     plot3(CoilPts(CrossedId(1),LR).Coord(1,Xid-1),...
%           CoilPts(CrossedId(1),LR).Coord(2,Xid-1),...
%           CoilPts(CrossedId(1),LR).Coord(3,Xid-1),'or');
else
    LR = 1;
    CrossedId = circshift(1:Stent.CoilNrOf,[0,+(XingNr-CNr)]);
%     Fac = 0.05;
end
%% Segment from the midline to the Xing => W3
%-----------------------------------------
OCSeg = CoilPts(CNr,RL).Coord(:,Xid) - FinalCAS.Origin(:,Xid);
W3    = OCSeg/norm(OCSeg);
%% W1 & W2
%-----------------------------------------
% Segment from the previous point to the next of this line
PNSeg1 = CoilPts(CNr,RL).Coord(:,Xid+1)-CoilPts(CNr,RL).Coord(:,Xid-1);
v1     = PNSeg1/norm(PNSeg1);
% Segment from the previous point to the next of this line
PNSeg2 = CoilPts(CrossedId(1),LR).Coord(:,Xid+1)-
CoilPts(CrossedId(1),LR).Coord(:,Xid-1);
v2    = PNSeg2/norm(PNSeg2);
% W1 & W3
W1 = cross(0.5*(v1+v2),W3)/norm(cross(0.5*(v1+v2),W3));
W2 = cross(W3,W1)/norm(cross(W1,W3));
% Angle
Alpha = acos(dot(v1,v2));
% Diameters
OD = 2*Stent.CRad*cos(Alpha/2)/sin(Alpha);
CD = 2*Stent.CRad*sin(Alpha/2)/sin(Alpha);
[RightPts1,LeftPts1,RightPts2,LeftPts2] = ...
    Ellypse(CoilPts(CNr,RL).Coord(:,Xid),W1,W2,W3,CD,OD,0);
if RL ==1
    EntryPts = RightPts1;
    ExitPts  = RightPts2;
else
    EntryPts = LeftPts1;
    ExitPts  = LeftPts2;
```

```
end
%% Plot
%------
if PlotVectors == 1
%     quiver3(CoilPts(CNr,RL).Coord(1,Xid),...
%             CoilPts(CNr,RL).Coord(2,Xid),...
%             CoilPts(CNr,RL).Coord(3,Xid),W1(1),W1(2),W1(3),Fac,'r')
%     quiver3(CoilPts(CNr,RL).Coord(1,Xid),...
%             CoilPts(CNr,RL).Coord(2,Xid),...
%             CoilPts(CNr,RL).Coord(3,Xid),W2(1),W2(2),W2(3),Fac,'g')
%     quiver3(CoilPts(CNr,RL).Coord(1,Xid),...
%             CoilPts(CNr,RL).Coord(2,Xid),...
%             CoilPts(CNr,RL).Coord(3,Xid),W3(1),W3(2),W3(3),Fac,'b')
    quiver3(CoilPts(CNr,RL).Coord(1,Xid),...
            CoilPts(CNr,RL).Coord(2,Xid),...
            CoilPts(CNr,RL).Coord(3,Xid),v1(1),v1(2),v1(3),Fac,'r')
    quiver3(CoilPts(CNr,RL).Coord(1,Xid),...
            CoilPts(CNr,RL).Coord(2,Xid),...
            CoilPts(CNr,RL).Coord(3,Xid),v2(1),v2(2),v2(3),Fac,'b')
    axis equal; grid on; axis vis3d

end
end
```

DoXingSegments.m

```
function [MeshPts,MeshCells,LMeshPts,RMeshPts] =...
    DoXingSegments(MeshPts,MeshCells,LCNr,i,LCoilFun,RCoilFun,...
    MidLinePts,LMeshPts,RMeshPts,LXingRegistry)

global Param
global Plot
% Right crossing strut
RCNr = LXingRegistry(i,LCNr);
% Number of existing mesh points
Cntr0 = size(MeshPts,1);
% Functions
LFun = LCoilFun{LCNr};
LDer = fnder(LFun,1);
RFun = RCoilFun{RCNr};
RDer = fnder(RFun,1);
% Mid line Points
MidPt  = MidLinePts(i,:);
MidPtb = MidLinePts(i-1,:);
MidPtn = MidLinePts(i+1,:);
% Prepare circle and ellypse points
v1 = ppval(LDer,LFun.breaks(i))/norm(ppval(LDer,LFun.breaks(i)));
v2 = ppval(RDer,RFun.breaks(i))/norm(ppval(RDer,RFun.breaks(i)));
[EPts,ePts] = DoCoilEllipses(v1,v2);
CPts = DoCoilCircles();
% Mesh points @ Xing
XPt    = ppval(LFun,LFun.breaks(i))';
Xw3    = (XPt-MidPt)'/norm(XPt-MidPt);
Xw1    = cross(0.5*(v1+v2),Xw3)/norm(cross(0.5*(v1+v2),Xw3));
Xw2    = cross(Xw3,Xw1)/norm(cross(Xw3,Xw1));
EMPts1 = repmat(XPt',1,Param.StrutsPtsNrOf)  + [Xw1,Xw3,Xw2]*EPts;
EMPts2 = repmat(XPt',1,Param.StrutsPtsNrOf)  + [Xw2,Xw3,Xw1]*ePts;
MeshPts(Cntr0+ 1:Cntr0+6,1:3) = EMPts2';
MeshPts(Cntr0+ 7:Cntr0+10,1:3) = EMPts1(:,[2,3,5,6])';
CntrLb = Cntr0+10;
% Mesh points @ Lbefore
if isnan(LMeshPts(LCNr,i-1,1))
    LPtb   = ppval(LFun,LFun.breaks(i-1))';
    Lbw3   = (LPtb-MidPtb)' / norm(LPtb-MidPtb);
    Lbw1   = ppval(LDer,LFun.breaks(i-1))/norm(ppval(LDer,LFun.breaks(i-
1)));
    Lbw2   = cross(Lbw3,Lbw1)/norm(cross(Lbw3,Lbw1));
    MPtsLb = repmat(LPtb',1,Param.StrutsPtsNrOf) + [Lbw3,Lbw2,Lbw1]*CPts;
    MeshPts(CntrLb+ 1:CntrLb+ 6,1:3) = MPtsLb';
    LMeshPts(LCNr,i-1,1:6) = CntrLb+ 1:CntrLb+ 6;
    CntrLn = CntrLb + 6;
else
    PtNr(1:6) = LMeshPts(LCNr,i-1,:);
    MPtsLb    = MeshPts(PtNr,:)';
    CntrLn    = CntrLb;
end
% Mesh points @ Lnext
if isnan(LMeshPts(LCNr,i+1,1))
    LPtn   = ppval(LFun,LFun.breaks(i+1))';
    Lnw3   = (LPtn-MidPtn)' / norm(LPtn-MidPtn);
```

```
    Lnw1   =
ppval(LDer,LFun.breaks(i+1))/norm(ppval(LDer,LFun.breaks(i+1)));
    Lnw2   = cross(Lnw3,Lnw1)/norm(cross(Lnw3,Lnw1));
    MPtsLn = repmat(LPtn',1,Param.StrutsPtsNrOf) + [Lnw3,Lnw2,Lnw1]*CPts;
    MeshPts(CntrLn+ 1:CntrLn+ 6,1:3) = MPtsLn';
    LMeshPts(LCNr,i+1,1:6) = CntrLn+ 1:CntrLn+ 6;
    CntrRb = CntrLn+6;
else
    PtNr(1:6) = LMeshPts(LCNr,i+1,:);
    MPtsLn    = MeshPts(PtNr,:)';
    CntrRb    = CntrLn;
end
% Mesh points @ Rbefore
if isnan(RMeshPts(RCNr,i-1,1))
    RPtb   = ppval(RFun,RFun.breaks(i-1))';
    Rbw3   = (RPtb-MidPtb)' / norm(RPtb-MidPtb);
    Rbw1   = ppval(RDer,RFun.breaks(i-1))/norm(ppval(RDer,RFun.breaks(i-
1)));
    Rbw2   = cross(Rbw3,Rbw1)/norm(cross(Rbw3,Rbw1));
    MPtsRb = repmat(RPtb',1,Param.StrutsPtsNrOf) + [Rbw3,Rbw2,Rbw1]*CPts;
    MeshPts(CntrRb+ 1:CntrRb+6,1:3) = MPtsRb';
    RMeshPts(RCNr,i-1,1:6) = CntrRb+ 1:CntrRb+6;
    CntrRn = CntrRb+6;
else
    PtNr(1:6) = RMeshPts(RCNr,i-1,:);
    MPtsRb    = MeshPts(PtNr,:)';
    CntrRn = CntrRb;
end
% Mesh points @ Rnext
if isnan(RMeshPts(RCNr,i+1,1))
    RPtn   = ppval(RFun,RFun.breaks(i+1))';
    Rnw3   = (RPtn-MidPtn)' / norm(RPtn-MidPtn);
    Rnw1   =
ppval(RDer,RFun.breaks(i+1))/norm(ppval(RDer,RFun.breaks(i+1)));
    Rnw2   = cross(Rnw3,Rnw1)/norm(cross(Rnw3,Rnw1));
    MPtsRn = repmat(RPtn',1,Param.StrutsPtsNrOf) + [Rnw3,Rnw2,Rnw1]*CPts;
    MeshPts(CntrRn+ 1:CntrRn+ 6,1:3) = MPtsRn';
    RMeshPts(RCNr,i+1,1:6) = CntrRn+ 1:CntrRn+ 6;
else
    PtNr(1:6) = RMeshPts(RCNr,i+1,:);
    MPtsRn    = MeshPts(PtNr,:)';
end
% Record Elements
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-1,1),LMeshPts(LCNr,i-
1,2),Cntr0+10];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-1,2),LMeshPts(LCNr,i-1,3),Cntr0+
1];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-1,3),LMeshPts(LCNr,i-1,4),Cntr0+
6];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-1,4),LMeshPts(LCNr,i-1,5),Cntr0+
5];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-1,5),LMeshPts(LCNr,i-1,6),Cntr0+
4];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-1,6),LMeshPts(LCNr,i-1,1),Cntr0+
9];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-1,1),Cntr0+10,Cntr0+ 9];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-1,2),Cntr0+ 1,Cntr0+10];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-1,3),Cntr0+ 6,Cntr0+ 1];
```

```
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-1,4),Cntr0+ 5,Cntr0+ 6];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-1,5),Cntr0+ 4,Cntr0+ 5];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i-1,6),Cntr0+ 9,Cntr0+ 4];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i+1,1),Cntr0+
2,LMeshPts(LCNr,i+1,2)];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i+1,2),Cntr0+
1,LMeshPts(LCNr,i+1,3)];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i+1,3),Cntr0+
7,LMeshPts(LCNr,i+1,4)];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i+1,4),Cntr0+
8,LMeshPts(LCNr,i+1,5)];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i+1,5),Cntr0+
4,LMeshPts(LCNr,i+1,6)];
MeshCells(end+1,1:3) = [LMeshPts(LCNr,i+1,6),Cntr0+
3,LMeshPts(LCNr,i+1,1)];
MeshCells(end+1,1:3) = [Cntr0+ 3,Cntr0+ 2,LMeshPts(LCNr,i+1, 1)];
MeshCells(end+1,1:3) = [Cntr0+ 2,Cntr0+ 1,LMeshPts(LCNr,i+1, 2)];
MeshCells(end+1,1:3) = [Cntr0+ 1,Cntr0+ 7,LMeshPts(LCNr,i+1, 3)];
MeshCells(end+1,1:3) = [Cntr0+ 7,Cntr0+ 8,LMeshPts(LCNr,i+1, 4)];
MeshCells(end+1,1:3) = [Cntr0+ 8,Cntr0+ 4,LMeshPts(LCNr,i+1, 5)];
MeshCells(end+1,1:3) = [Cntr0+ 4,Cntr0+ 3,LMeshPts(LCNr,i+1, 6)];
MeshCells(end+1,1:3) = [Cntr0+ 6,Cntr0+ 5,RMeshPts(RCNr,i-1,1)];
MeshCells(end+1,1:3) = [Cntr0+ 1,Cntr0+ 6,RMeshPts(RCNr,i-1,2)];
MeshCells(end+1,1:3) = [Cntr0+ 7,Cntr0+ 1,RMeshPts(RCNr,i-1,3)];
MeshCells(end+1,1:3) = [Cntr0+ 8,Cntr0+ 7,RMeshPts(RCNr,i-1,4)];
MeshCells(end+1,1:3) = [Cntr0+ 4,Cntr0+ 8,RMeshPts(RCNr,i-1,5)];
MeshCells(end+1,1:3) = [Cntr0+ 5,Cntr0+ 4,RMeshPts(RCNr,i-1,6)];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i-1,4),RMeshPts(RCNr,i-1,5),Cntr0+
8];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i-1,5),RMeshPts(RCNr,i-1,6),Cntr0+
4];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i-1,6),RMeshPts(RCNr,i-1,1),Cntr0+
5];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i-1,1),RMeshPts(RCNr,i-1,2),Cntr0+
6];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i-1,2),RMeshPts(RCNr,i-1,3),Cntr0+
1];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i-1,3),RMeshPts(RCNr,i-1,4),Cntr0+
7];
MeshCells(end+1,1:3) =
[RMeshPts(RCNr,i+1,2),RMeshPts(RCNr,i+1,1),Cntr0+10];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i+1,3),RMeshPts(RCNr,i+1,2),Cntr0+
1];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i+1,4),RMeshPts(RCNr,i+1,3),Cntr0+
2];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i+1,5),RMeshPts(RCNr,i+1,4),Cntr0+
3];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i+1,6),RMeshPts(RCNr,i+1,5),Cntr0+
4];
MeshCells(end+1,1:3) = [RMeshPts(RCNr,i+1,1),RMeshPts(RCNr,i+1,6),Cntr0+
9];
MeshCells(end+1,1:3) = [Cntr0+ 9,Cntr0+10,RMeshPts(RCNr,i+1,1)];
MeshCells(end+1,1:3) = [Cntr0+10,Cntr0+ 1,RMeshPts(RCNr,i+1,2)];
MeshCells(end+1,1:3) = [Cntr0+ 1,Cntr0+ 2,RMeshPts(RCNr,i+1,3)];
MeshCells(end+1,1:3) = [Cntr0+ 2,Cntr0+ 3,RMeshPts(RCNr,i+1,4)];
MeshCells(end+1,1:3) = [Cntr0+ 3,Cntr0+ 4,RMeshPts(RCNr,i+1,5)];
MeshCells(end+1,1:3) = [Cntr0+ 4,Cntr0+ 9,RMeshPts(RCNr,i+1,6)];
```

```
% Visualisation
%--------------
% Mesh
if Plot.Vertices==1
    hold on
    plot3(MPtsLb(1,[1:end,1]),MPtsLb(2,[1:end,1]),MPtsLb(3,[1:end,1]),'.-
m')
    plot3(EMPts2(1,[1:end,1]),EMPts2(2,[1:end,1]),EMPts2(3,[1:end,1]),'.-
m')
    plot3(MPtsLn(1,[1:end,1]),MPtsLn(2,[1:end,1]),MPtsLn(3,[1:end,1]),'.-
m')
    plot3(MPtsRb(1,[1:end,1]),MPtsRb(2,[1:end,1]),MPtsRb(3,[1:end,1]),'.-
m')
    plot3(EMPts1(1,[1:end,1]),EMPts1(2,[1:end,1]),EMPts1(3,[1:end,1]),'.-
m')
    plot3(MPtsRn(1,[1:end,1]),MPtsRn(2,[1:end,1]),MPtsRn(3,[1:end,1]),'.-
m')
    axis equal; grid on; box on;
end
if Plot.Vectors == 1
    %  vectors
quiver3(XPt(1),XPt(2),XPt(3),v1(1),v1(2),v1(3),0.1,'Color','b','LineWidth
',2)
quiver3(XPt(1),XPt(2),XPt(3),v2(1),v2(2),v2(3),0.1,'Color','b','LineWidth
',2)
    hold on
quiver3(XPt(1),XPt(2),XPt(3),Xw1(1),Xw1(2),Xw1(3),0.1,'Color','c','LineWi
dth',2)
quiver3(XPt(1),XPt(2),XPt(3),Xw2(1),Xw2(2),Xw2(3),0.1,'Color','c','LineWi
dth',2)
quiver3(XPt(1),XPt(2),XPt(3),Xw3(1),Xw3(2),Xw3(3),0.1,'Color','c','LineWi
dth',2)
quiver3(LPtn(1),LPtn(2),LPtn(3),Lnw3(1),Lnw3(2),Lnw3(3),0.1,'Color','r','
LineWidth',2)
quiver3(LPtn(1),LPtn(2),LPtn(3),Lnw1(1),Lnw1(2),Lnw1(3),0.1,'Color','r','
LineWidth',2)
quiver3(LPtn(1),LPtn(2),LPtn(3),Lnw2(1),Lnw2(2),Lnw2(3),0.1,'Color','r','
LineWidth',2)
quiver3(LPtb(1),LPtb(2),LPtb(3),Lbw3(1),Lbw3(2),Lbw3(3),0.1,'Color','r','
LineWidth',2)
quiver3(LPtb(1),LPtb(2),LPtb(3),Lbw1(1),Lbw1(2),Lbw1(3),0.1,'Color','r','
LineWidth',2)
quiver3(LPtb(1),LPtb(2),LPtb(3),Lbw2(1),Lbw2(2),Lbw2(3),0.1,'Color','r','
LineWidth',2)
quiver3(RPtn(1),RPtn(2),RPtn(3),Rnw3(1),Rnw3(2),Rnw3(3),0.1,'Color','r','
LineWidth',2)
quiver3(RPtn(1),RPtn(2),RPtn(3),Rnw1(1),Rnw1(2),Rnw1(3),0.1,'Color','r','
LineWidth',2)
quiver3(RPtn(1),RPtn(2),RPtn(3),Rnw2(1),Rnw2(2),Rnw2(3),0.1,'Color','r','
LineWidth',2)
quiver3(RPtb(1),RPtb(2),RPtb(3),Rbw3(1),Rbw3(2),Rbw3(3),0.1,'Color','r','
LineWidth',2)
quiver3(RPtb(1),RPtb(2),RPtb(3),Rbw1(1),Rbw1(2),Rbw1(3),0.1,'Color','r','
LineWidth',2)
quiver3(RPtb(1),RPtb(2),RPtb(3),Rbw2(1),Rbw2(2),Rbw2(3),0.1,'Color','r','
LineWidth',2)
    axis equal; box on; grid on;
end
```

```
% points
if Plot.XingHelpPts==1
    plot3(XPt(1),XPt(2),XPt(3),'*k')
    hold on
    plot3(LPtn(1),LPtn(2),LPtn(3),'.b')
    plot3(LPtb(1),LPtb(2),LPtb(3),'.b')
    plot3(RPtn(1),RPtn(2),RPtn(3),'.r')
    plot3(RPtb(1),RPtb(2),RPtb(3),'.r')
    plot3(MidPt(1),MidPt(2),MidPt(3),'.m')
    plot3(MidPtb(1),MidPtb(2),MidPtb(3),'.m')
    plot3(MidPtn(1),MidPtn(2),MidPtn(3),'.m')
    axis equal; grid on; box on;
end
```

Ellypse.m

```
function [RightPts1,LeftPts1,RightPts2,LeftPts2] = ...
    Ellypse(Orig,E1,E2,E3,d2,d3,PlotFlag)
global Stent
    tt  = linspace(0,2*pi,7);
    Pts1 = [zeros(1,length(tt)-1);
           d3*sin(tt(1:end-1));
           Stent.CRad*cos(tt(1:end-1))];
    Pts2 = [zeros(1,length(tt)-1);
           d2*sin(tt(1:end-1));
           Stent.CRad*cos(tt(1:end-1))];
    TrafoMtx1 = [E1,E2,E3];
    TrafoMtx2 = [E2,E1,E3];
    EPts1 = TrafoMtx1*Pts1+repmat(Orig,1,length(tt)-1);
    EPts2 = TrafoMtx2*Pts2+repmat(Orig,1,length(tt)-1);
    if PlotFlag == 1
        plot3(EPts1(1,:),EPts1(2,:),EPts1(3,:),'.-r')
        hold on
        plot3(EPts2(1,:),EPts2(2,:),EPts2(3,:),'.-b')
        axis equal; grid on; box on;
    end
    RightPts1  = [EPts1(:,[1,6:-1:4]),EPts2(:,5:6)];%
%     plot3(RightPts1(1,:),RightPts1(2,:),RightPts1(3,:),'og'); hold on;
    LeftPts1 = [EPts2(:,1:4),EPts1(:,5:6)];%
%     plot3(LeftPts1(1,:),LeftPts1(2,:),LeftPts1(3,:),'og'); hold on;
    RightPts2  = [EPts2(:,1:4),EPts1(:,3:-1:2)];
%     plot3(RightPts2(1,:),RightPts2(2,:),RightPts2(3,:),'og'); hold on;
    LeftPts2 = [EPts1(:,1:4),EPts2(:,5:6)];%
%     plot3(LeftPts2(1,:),LeftPts2(2,:),LeftPts2(3,:),'og'); hold on;
end
```

ErrorFun.m

```
function Output =
ErrorFun(SlicesForOptim,DPosInput,GoalRValues,InitGuess,PlotFlag)
global FigForError
global NomCAS
global ErrorPlot
global GlobRadDiff0
global PosDependentCurvature0
DPos = DPosInput;
% Actualise the plot
%-------------------
if PlotFlag==1
    % Create the Dislplaced points
    for i = 1:length(SlicesForOptim)
        Slice = SlicesForOptim(i);
        Orig     = NomCAS.Origin(Slice,:);
        TrafoMtx = [NomCAS.E1(Slice,:)/norm(NomCAS.E1(Slice,:));...
                    NomCAS.E2(Slice,:)/norm(NomCAS.E2(Slice,:));...
                    NomCAS.E3(Slice,:)/norm(NomCAS.E3(Slice,:))];
        NewCLinePts(i,:)  = (TrafoMtx\([0;DPos(:,i)])+Orig')';
    end
    % plot the new points
    figure(FigForError)
    set(ErrorPlot,'XData',NewCLinePts(:,1));
    set(ErrorPlot,'YData',NewCLinePts(:,2));
    set(ErrorPlot,'ZData',NewCLinePts(:,3));
    drawnow;
end
% Save the first errors for normalisation (only on the first run)
%----------------------------------------------------------------
if isnan(GlobRadDiff0)
    GlobRadDiff0            = GlobRadDiff(SlicesForOptim,...
        InitGuess,GoalRValues,0);
    PosDependentCurvature0  = PosDependentCurvature(SlicesForOptim,...
        InitGuess,GoalRValues,0);
end
Output =
1*(GlobRadDiff(SlicesForOptim,DPos,GoalRValues,0)/GlobRadDiff0)+...
    ...
    1*(PosDependentCurvature(SlicesForOptim,DPos,GoalRValues,0)/...
    PosDependentCurvature0);
```

FindNewCenterOfMass.m

```
function [NewCLinePts,RadPtsOptim] =
FindNewCenterOfMass(GoalRValues,PlotFlag)
disp('FindNewCenterOfMass...')
%% Global Values
%--------------------------------------------------------------------------
global NomCAS
global Param
global Name
global OptDPos
global GlobRadDiff0;
global FigForError
global ErrorPlot
%% Looks if a Flag File exists
%--------------------------------------------------------------------------
if exist([Name.StlFile_Name,'_6b_OptCLinePts.mat']) &&...
   exist([Name.StlFile_Name,'_6c_RadPtsOptim.mat'])
    load([Name.StlFile_Name,'_6b_OptCLinePts.mat'],'NewCLinePts')
    load([Name.StlFile_Name,'_6c_RadPtsOptim.mat'],'RadPtsOptim')
    disp(['=> ',Name.StlFile_Name,'_6b_OptCLinePts.mat',' has been
loaded'])
    disp(['=> ',Name.StlFile_Name,'_6c_RadPtsOptim.mat',' has been
loaded'])
    Plot(NomCAS,NewCLinePts,PlotFlag)
    return
end
% Start the figure for the visualisation of the optimisation
%--------------------------------------------------------------------------
if PlotFlag==Param.OptimVisualisationFlag
    FigForError = figure('Position',Param.FigPos,...
        'Name','Optimisation Visualisation',...
        'NumberTitle','off');
    plot3(NomCAS.Origin(1:Param.OptimSlices(1)-1,1),...
          NomCAS.Origin(1:Param.OptimSlices(1)-1,2),...
          NomCAS.Origin(1:Param.OptimSlices(1)-1,3),'.-r');
    hold on;
    ErrorPlot = plot3(NomCAS.Origin(Param.OptimSlices,1),...
          NomCAS.Origin(Param.OptimSlices,2),...
          NomCAS.Origin(Param.OptimSlices,3),'.-b');
    plot3(NomCAS.Origin(Param.OptimSlices,1),...
          NomCAS.Origin(Param.OptimSlices,2),...
          NomCAS.Origin(Param.OptimSlices,3),'-g')
    plot3(NomCAS.Origin(Param.OptimSlices(end)+1:end,1),...
          NomCAS.Origin(Param.OptimSlices(end)+1:end,2),...
          NomCAS.Origin(Param.OptimSlices(end)+1:end,3),'.-r')
    grid on; box on; axis equal;
    legend({'Fix Centerline','Centerline Under Optimisation',...
        'Original Centerline'});
    drawnow;
end
% Prepare optimisation
%--------------------------------------------------------------------------
InitGuess = zeros(2,size(Param.OptimSlices,2));
options = optimset('MaxFunEvals',1e7,'MaxIter',1e4,'Display','iter',...
    'TolFun',1e-3,'TolX',1e-3);
```

```
% options = optimset('MaxFunEvals',200,'MaxIter',200,'Display','iter',...
% 'TolFun',1e-3,'TolX',1e-3);
% Preoptimisation
%-------------------------------------------------------------------------
for NN = 2.^(Param.PreOptimRunNrOf:-1:1)
    disp(['Preoptimisation with NN = ',num2str(NN)])
    GlobRadDiff0 = NaN; % Tells the error function to remember the
result0
    [OptDPos, ValueError]  = fminunc(@(DPos) ErrorFun(...
        Param.OptimSlices(1:NN:end),DPos,GoalRValues,...
        InitGuess(:,(1:NN:end)),Param.OptimVisualisationFlag),...
        InitGuess(:,1:NN:end),options);
    % replace the values of the initial guess with thos that have already
    % been optimised
    InitGuess(:,1:NN:end) = OptDPos;
    % Prepare the next initial guess
    j = 1+NN/2;
    for i=1:size(OptDPos(:,1:end-1),2)
        InitGuess(:,j) = 0.5*(OptDPos(:,i)+OptDPos(:,i+1));
        j = j+NN;
    end
end
% Optimisation
%-------------------------------------------------------------------------
% FirstRun     = 1;
GlobRadDiff0 = NaN;
NN = 1;
[OptDPos, ValueError]  = fminunc(@(DPos) ErrorFun(...
    Param.OptimSlices(1:NN:end),DPos,GoalRValues,...
    InitGuess(:,(1:NN:end)),Param.OptimVisualisationFlag),...
    InitGuess(:,1:NN:end),options);
% Prepare the Outputs
%-------------------------------------------------------------------------
% Calculate the new centerline
%-----------------------------
NewCLinePts = NomCAS.Origin;
for i = 1:length(Param.OptimSlices)
    Slice = Param.OptimSlices(i);
    Orig     = NomCAS.Origin(Slice,:);
    TrafoMtx = [NomCAS.E1(Slice,:)/norm(NomCAS.E1(Slice,:));...
                NomCAS.E2(Slice,:)/norm(NomCAS.E2(Slice,:));...
                NomCAS.E3(Slice,:)/norm(NomCAS.E3(Slice,:))];
    NewCLinePts(Slice,:)  = (TrafoMtx\([0;OptDPos(:,i)])+Orig')';
end
% Calculate the new WallMaxRadius
%--------------------------------
Rad = PosDependentRadius(Param.OptimSlices,OptDPos,0);
RadPtsOptim = GoalRValues;
RadPtsOptim(Param.OptimSlices) = Rad;
% Plot and save the results
%-------------------------------------------------------------------------
Plot(NomCAS,NewCLinePts,PlotFlag)
% save([Name.StlFile_Name,'_6a_OptCLineFun.mat'],'OptCLineFun')
save([Name.StlFile_Name,'_6b_OptCLinePts.mat'],'NewCLinePts')
save([Name.StlFile_Name,'_6c_RadPtsOptim.mat'],'RadPtsOptim')
end
```


```
function Plot(NomCAS,NewCLinePts,PlotFlag)
global Param
%% Plot the NewCoM, just for check:
%--------------------------------------------------------------------------
if PlotFlag == 1
    figure('Position',Param.FigPos,...
        'Name','FindNewCenterOfMass: Results of the optimisation',...
        'NumberTitle','off')
    plot3(NomCAS.Origin(:,1),NomCAS.Origin(:,2),NomCAS.Origin(:,3),'o-
b');
    hold on
    plot3(NewCLinePts(:,1),NewCLinePts(:,2),NewCLinePts(:,3),'*-r');
    legend({'Old CLine Pts','New CLine Pts'})
    grid on; box on; axis equal;
end
end
```

FlagFace.m

```
function [GlobalFaceFlag] = FlagFace()
% function [GlobalFaceFlag,FaceNormId] = FlagFace(F,V,N,FigH,SurfH)
disp('Flag Face...')
%% Global Values
%--------------------------------------------------------------------------
global FaceNormH
global Param
global Name
global F; global V; global N;
%% Looks if a Flag File exists
%--------------------------------------------------------------------------
if exist([Name.StlFile_Name,'_1_GlobalFaceFlag.mat'])
    load([Name.StlFile_Name,'_1_GlobalFaceFlag.mat'],'GlobalFaceFlag')
    disp(['=> ',Name.StlFile_Name,'_1_GlobalFaceFlag.mat',' has been
loaded'])
    return
end
% Create the Figure of the vesssel
H.VesselFig.Fig  = figure('Name','FlagFace','Position',Param.FigPos,...
    'NumberTitle','off');
CData            = zeros(1,size(F,1)); % Color Data
H.VesselFig.Surf = trisurf(F,V(:,1),V(:,2),V(:,3),CData);
axis equal; axis vis3d;
% Initialisation
%---------------
FaceNr         = 0;
DoFaceFlag     = 1;
GlobalFaceFlag = zeros(size(F,1),1);
while DoFaceFlag == 1
    % Actulisation / Initinalisation
    %-------------------------------
    FaceNr   = FaceNr + 1;
    FaceFlag = zeros(size(F,1),1);
    % Adapt the plot
    %---------------
    set(H.VesselFig.Surf,'CData',FaceFlag)   % Uniform color for all
elements
    if exist('FaceNormH','var')   % Remove face normal (if present)
        if ishandle(FaceNormH); delete(FaceNormH); end
    end
    % Select the candidate triangles and adjust the color of their faces
    %-------------------------------------------------------------------
    set(H.VesselFig.Fig,'Name',['Select Face ',num2str(FaceNr)]);
    CandFlagVec         = SelectMeshTri(H.VesselFig.Fig);
    CandTriId           = find(CandFlagVec==1);
    NotCandTriId        = find(CandFlagVec==0);
    CData(CandTriId)    = 1; CData(NotCandTriId) = 0;
    set(H.VesselFig.Surf,'CData',CData)
    % Select the triangle for the normal, adjust
    % the color of its face and plot its normal
    %------------------------------------------
    set(H.VesselFig.Fig,'Name',['Select Triangle for Normal
',num2str(FaceNr)]);
    NormTriId        = SelectOneTri(H.VesselFig.Fig);
    CData(NormTriId) = 0.5;
```

```
    % Plot the its Normal
    TriCentroid =
[mean(V(F(NormTriId,:),1)),mean(V(F(NormTriId,:),2)),...
                  mean(V(F(NormTriId,:),3))];
    A1          = N(NormTriId,:); % Normal vector
    hold on
    FaceNormH   =
quiver3(TriCentroid(1),TriCentroid(2),TriCentroid(3),...
                       A1(1),A1(2),A1(3));
    % Search for the triangles of the face and change its color
    %-------------------------------------------------------------
    for j=1:size(CandTriId,1)
        if abs(dot(A1,N(CandTriId(j),:))) >
cos(Param.AngleForFace*pi/180)
            FaceFlag(CandTriId(j))=1;
        end
    end
    FaceTriId           = find(FaceFlag==1);
    NotFaceTriId        = find(FaceFlag==0);
    CData(FaceTriId)    = 1;
    CData(NotFaceTriId) = 0;
    set(H.VesselFig.Surf,'CData',CData);
    FaceNormId(FaceNr) = NormTriId;
    GlobalFaceFlag(find(FaceFlag)) = FaceNr;
    % Decide What to do next
    %-----------------------
    Ans = questdlg('What now?','What to do next',...
                   'Reflag this face','Flag Next face',...
                   'Done Flagging','Flag Next face');
    if strcmp(Ans,'Done Flagging')
        break
    elseif strcmp(Ans,'Reflag this face')
        FaceNr = FaceNr-1;
    end
end
save([Name.StlFile_Name,'_1_GlobalFaceFlag'],'GlobalFaceFlag')
% Create the Figure of the vesssel
close(H.VesselFig.Fig);
```

FlagLines.m

```
function LinePtFlag = FlagLines(GlobalFaceFlag)
%% Global Values
%--------------------------------------------------------------------------
global F; global V;
LinePtFlag = zeros(size(V,1),1);
for i=1:length(unique(GlobalFaceFlag))-1
    FacePtsIdMtxForm = F(find(GlobalFaceFlag==i),:);
    FacePtsIdRowForm = unique(reshape(FacePtsIdMtxForm,1,[]));
    FacePts          = V(FacePtsIdRowForm(:),:);
    WallPtsIdMtxForm = F(find(GlobalFaceFlag==0),:);
    WallPtsIdRawForm = unique(reshape(WallPtsIdMtxForm,1,[]));
    WallPts          = V(WallPtsIdRawForm(:),:);
    IntsctFaceFlag = ismember(FacePts,WallPts,'rows');
    IntsctPtsId    = FacePtsIdRowForm(find(IntsctFaceFlag~=0));
    LinePtFlag(IntsctPtsId) = i;
end
```

FlagPtsBetwWalls.m

```
function Flag = FlagPtsBetwWalls(x0ik,WallPts,varargin)
disp('FlagPtsBetwWalls...')
% Handle the varargin
%--------------------
if isempty(varargin)
    ShowWaitBarFlag = 0;
else
    ShowWaitBarFlag = 1;
end
%% Points Selection
%--------------------------------------------------------------------------
% Numbering Maxima
%-----------------
pMax = size(WallPts,1); % Number of plane limitating the area
iMax = size(x0ik,1);    % Number of points
% Prepare the ray vectors v
%--------------------------
% The 4 chosen direction have to be perpendiculat to the depth direction
% The First direction will be any direction perp to the depth vector
DepthDir(1,1:3) = (WallPts(1,2,1:3)-WallPts(1,1,1:3));
Dir1 = [NaN,NaN,NaN];
Tmp  = 0;
% Small trick to compesate for the most improbable case where the first
% testted vector could be parallel to the DepthDir
while any(isnan(Dir1))
    Tmp=Tmp+1;
    Dir1(1,1:3) = cross([2-Tmp,1+Tmp,1],DepthDir)/...
                  norm(cross([2-Tmp,1+Tmp,1],DepthDir));
end
% The second direction is the crosspd(Dir1,DepthDirection)
Dir2 = cross(DepthDir,Dir1)/norm(cross(DepthDir,Dir1));
% The third direction is the mean of Dir1 and Dir2
Dir3 = mean([Dir1;Dir2])/norm(mean([Dir1;Dir2]));
vdk = [ Dir1 ; Dir2 ; Dir3 ];
% Number of directions
dMax = size(vdk,1);
% Wall Corner Points
%-------------------
x1pk = zeros(pMax,3);
x4pk = zeros(pMax,3);
FaceMidPtpk = zeros(pMax,3);
npk  = zeros(pMax,3);
for p=1:pMax
    % Prepare the  the plane points
    %------------------------------
    x1pk(p,1:3) = WallPts(p,1,1:3);
    x4pk(p,1:3) = WallPts(p,2,1:3);
end
x2pk = circshift(x1pk,[-1,0]);
x3pk = circshift(x4pk,[-1,0]);
WallXmin = min([WallPts(:,1,1);WallPts(:,2,1)]);
WallXmax = max([WallPts(:,1,1);WallPts(:,2,1)]);
WallYmin = min([WallPts(:,1,2);WallPts(:,2,2)]);
WallYmax = max([WallPts(:,1,2);WallPts(:,2,2)]);
WallZmin = min([WallPts(:,1,3);WallPts(:,2,3)]);
```

```
WallZmax = max([WallPts(:,1,3);WallPts(:,2,3)]);
% Prepare the Wall Middle point and normals
%------------------------------
for p = 1:pMax
    FaceMidPtpk(p,:) = mean([x1pk(p,:);x2pk(p,:);x3pk(p,:);x4pk(p,:)],1);
    npk(p,1:3) = cross((x1pk(p,1:3)-x2pk(p,1:3)),(x3pk(p,1:3)-
x2pk(p,1:3)))/...
            norm(cross((x1pk(p,1:3)-x2pk(p,1:3)),(x3pk(p,1:3)-
x2pk(p,1:3))));
end
% %% Checks ( uncomment to plot many features)
% %-------------------------------------------------------------------------
% % Check the centroid
% %--------------------
% Centroid(1:3) = mean([WallPts(:,1,:);WallPts(:,2,:)]);
% plot3(Centroid(1),Centroid(2),Centroid(3),'sqc')
% hold on
% Colors = {'r','b','b','b'};
% Plane = 1;
% for p=1:pMax
%     if p==Plane
%         Color   = 'r';
%         MarkerC = 'o';
%         MarkerM = '*';
%     else
%         Color = 'b';
%         MarkerC = '.';
%         MarkerM = '.';
%     end
%     % Check Points
%     %-------------
%     text(WallPts(p,1,1),WallPts(p,1,2),...
%          WallPts(p,1,3),['(',num2str(p),',1,:)'])
%     text(WallPts(p,2,1),WallPts(p,2,2),...
%          WallPts(p,2,3),['(',num2str(p),',2,:)'])
%     plot3([x1pk(p,1),x2pk(p,1),x3pk(p,1),x4pk(p,1),x1pk(p,1)],...
%           [x1pk(p,2),x2pk(p,2),x3pk(p,2),x4pk(p,2),x1pk(p,2)],...
%           [x1pk(p,3),x2pk(p,3),x3pk(p,3),x4pk(p,3),x1pk(p,3)],...
%           'Color',Color,'Marker',MarkerC)
%     hold on
%     % Check Mid points
%     %-----------------
%     plot3(FaceMidPtpk(p,1),FaceMidPtpk(p,2),FaceMidPtpk(p,3),...
%          'Color',Color,'Marker',MarkerM)
%     % Check the plane normal vector
%     %------------------------------
%     quiver3(FaceMidPtpk(p,1),FaceMidPtpk(p,2),FaceMidPtpk(p,3),...
%         npk(p,1),npk(p,2),npk(p,3),5,'color',Color)
% end
% for d=1:dMax
%     % Check the directions
%     %---------------------
quiver3(Centroid(1),Centroid(2),Centroid(3),Dir1(1),Dir1(2),Dir1(3),...
%         5,'color','r')
quiver3(Centroid(1),Centroid(2),Centroid(3),Dir2(1),Dir2(2),Dir2(3),...
%         5,'color','g')
quiver3(Centroid(1),Centroid(2),Centroid(3),Dir3(1),Dir3(2),Dir3(3),...
%         5,'color','b')
```

```
% end
% % Check the x0 points
% %----------------------
% for i=1
%
plot3(V(F(i,[1:end,1]),1),V(F(i,[1:end,1]),2),V(F(i,[1:end,1]),3),'-m')
%     hold on
%     plot3(x0ik(i,1),x0ik(i,2),x0ik(i,3),'*m');
% end
% axis equal; grid on;
Flag = NaN*ones(iMax,1);
% Exclude all obvious cases
%--------------------------
ExcludedFlag = sum([x0ik(:,1)<WallXmin,x0ik(:,1)>WallXmax,...
                    x0ik(:,2)<WallYmin,x0ik(:,2)>WallYmax,...
                    x0ik(:,3)<WallZmin,x0ik(:,3)>WallZmax],2);
Flag(find(ExcludedFlag>0))=0;
if ShowWaitBarFlag==1
    WtBarH = waitbar(0,'Calculating who is inside/outside...') ;
else
    WtBarH = waitbar(0,'Plaese wait...','Visible','off');
end
NotExcludedIds = find(ExcludedFlag==0);
for iNotEx=1:length(NotExcludedIds)
    i = NotExcludedIds(iNotEx);
    waitbar(iNotEx/length(NotExcludedIds))
    DirNrOf = 0;
    for d=1:dMax
        XingNr = 0;
        for p=1:pMax
            t = dot(npk(p,:),(x1pk(p,:)-
x0ik(i,:)))/dot(npk(p,:),vdk(d,:));
            if t>0
                % Intersection point
                xzpid = x0ik(i,:)+vdk(d,:)*t;
                % Turning Directions
                CrossPdInt2Int3 = cross((x2pk(p,:)-xzpid),(x3pk(p,:)-
xzpid));
                CrossPdInt4Int1 = cross((x4pk(p,:)-xzpid),(x1pk(p,:)-
xzpid));
                % Check if the point is inside the limits
                if dot(npk(p,:),CrossPdInt2Int3)<0
                    if dot(npk(p,:),CrossPdInt4Int1)<0
                        XingNr = XingNr + 1;
                    end
                end
            end
        end
        % Uneven number of crossings means inside
        if mod(XingNr,2)==1
            DirNrOf = DirNrOf + 1;
        end
    end
    % if at least two of the tree directions indicate "inside" then it is
    % considered as "inside"
    if DirNrOf>=2
        Flag(i) = 1;
    else
```

```
        Flag(i) = 0;
    end
end
close(WtBarH)
```

GetCoM.m

```
function [CoM,BBFlags] = GetCoM(GlobalFaceFlag,PlotFlag)
disp('GetCoM...');
%% Global Values
%--------------------------------------------------------------------------
global Param;
global Name
global F; global V; global N;
%% Looks if the CoM-file existe
%--------------------------------------------------------------------------
if exist([Name.StlFile_Name,'_2_CoM.mat'])
    load([Name.StlFile_Name,'_2_CoM.mat'],'CoM','BBFlags')
    disp(['=> ',Name.StlFile_Name,'_2_CoM.mat',' has been loaded'])
    % Plot the loaded CoM
    if PlotFlag == 1
        [VesselFig,NewCoMH,AllCoMH] = InitiateFigure();
        plot3(CoM(:,1),CoM(:,2),CoM(:,3),'sqb',...
              'MarkerFaceColor','b');
    end
    return
end
[VesselFig,NewCoMH,AllCoMH] = InitiateFigure();
%% Loop the Bounding Box
%--------------------------------------------------------------------------
% Initialisation
Answer  = 'Do Next BB';
CoM     = [];
BBFlags = zeros(1,size(F,1));
while strcmp(Answer,'Do Next BB')
    % Select the bounding box
    [NewBBFlags,WallPts] = SelectMeshTriRect(F,V,VesselFig);
    % Calculate the New Centers of Mass
    NewCoM = CoMWithMapping(F,V,N,NewBBFlags,GlobalFaceFlag,WallPts)';
    % Plot the New CoM
    set(NewCoMH,'XData',NewCoM(:,1),'YData',NewCoM(:,2),...
        'ZData',NewCoM(:,3))
    % Question dlg
    Answer = questdlg({'What to do?'},'Question','Redo Last Step',...
        'Do Next BB','Done','Do Next BB');
    % Add the NewCoM to the CoM
    if not(strcmp(Answer,'Redo Last Step'))
        % Register the accepted CoM
        CoM = [CoM;NewCoM];
        % Register the accepted BBFlags
        BBFlags(find(NewBBFlags)) = 1;
        % Plot the accepted CoM
```


```
        set(AllCoMH,'XData',CoM(:,1),'YData',CoM(:,2),'ZData',CoM(:,3))
        % Erase the New CoM
        set(NewCoMH,'XData',NaN,'YData',NaN,'ZData',NaN)
    elseif strcmp(Answer,'Redo Last Step')
        % Change the answer to reenter the while loop
        Answer = 'Do Next BB';
        % Erase the New CoM
        set(NewCoMH,'XData',NaN,'YData',NaN,'ZData',NaN)
    end
end
%% Get the CoM for the covers
%--------------------------------------------------------------------------
for i=1:length(unique(GlobalFaceFlag))-1
    CoMCover(i,:) = mean(V(F(find(GlobalFaceFlag==i)),:));
end
% Register the CoM for the covers
CoM = [CoMCover(1,:);CoM;CoMCover(2,:)];
% Plot the CoM for the covers
set(AllCoMH,'XData',CoM(:,1),'YData',CoM(:,2),'ZData',CoM(:,3))
%% Save the CoM into a mat-file
%--------------------------------------------------------------------------
save([Name.StlFile_Name,'_2_CoM.mat'],'CoM','BBFlags')
uiwait(VesselFig,Param.WaitTimeout)
close(VesselFig)
end
function [VesselFig,NewCoMH,AllCoMH] = InitiateFigure()
%% Create the figure for the vessel and its CoM
%--------------------------------------------------------------------------
%% Global Values
%--------------------------------------------------------------------------
global Param;
global F; global V;
VesselFig = figure('Position',Param.FigPos,'Name',...
    'GetCoM: Choose the CoM Box','NumberTitle','off');
trisurf(F,V(:,1),V(:,2),V(:,3),0,'FaceAlpha',0.2);
hold on;
NewCoMH     = plot3(NaN,NaN,NaN,'sqb','MarkerFaceColor','b');
AllCoMH     = plot3(NaN,NaN,NaN,'sqr','MarkerFaceColor','r');
axis equal; axis vis3d;
end
```

GetCurvature.m

```
function Rc = GetCurvature(OptCLineFun)
disp('GetCurvature...');
NN = 10;
CLPts = ppval(OptCLineFun,linspace(OptCLineFun.breaks(1),...
    OptCLineFun.breaks(end),NN*(length(OptCLineFun.breaks)-1)+1));
% Values in colum:
x = CLPts(1,:);
y = CLPts(2,:);
z = CLPts(3,:);
dx  = gradient(x);
ddx = gradient(dx);
dy  = gradient(y);
ddy = gradient(dy);
dz  = gradient(z);
ddz = gradient(dz);
k = sqrt((ddz.*dy-ddy.*dz).^2+...
         (ddx.*dz-ddz.*dx).^2+...
         (ddy.*dx-ddx.*dy).^2)./...
         ((dx.^2+dy.^2+dz.^2).^(3/2));
Rc = k.^-1;
Rc = Rc(1:NN:end);
end
```

GetInitCurvature.m

```
function Rc = GetInitCurvature(NomCAS)
disp('GetCurvature...');
% Values in colum:
x = NomCAS.Origin(:,1);
y = NomCAS.Origin(:,2);
z = NomCAS.Origin(:,3);
dx  = gradient(x);
ddx = gradient(dx);
dy  = gradient(y);
ddy = gradient(dy);
dz  = gradient(z);
ddz = gradient(dz);
k = sqrt((ddz.*dy-ddy.*dz).^2+...
         (ddx.*dz-ddz.*dx).^2+...
         (ddy.*dx-ddx.*dy).^2)./...
         ((dx.^2+dy.^2+dz.^2).^(3/2));
Rc = (k.^-1)';
end
```

GetNominalCAS.m

```
function GetNominalCAS(CLineFun,PlotFlag)
disp('GetNominalCAS...')
%% global values
%-------------------------------------------------------------------------
global NomCAS
global Param
global V; global F;
global Name;
%% See if the variables exists
%-------------------------------------------------------------------------
if exist([Name.StlFile_Name,'_4_NomCAS.mat'],'file')
    Temp = load([Name.StlFile_Name,'_4_NomCAS.mat'],'NomCAS');
    if isfield(Temp.NomCAS,'TimeAlongCline') &&...
            isfield(Temp.NomCAS,'LengthAlongCline') &&...
            isfield(Temp.NomCAS,'Origin') &&...
            isfield(Temp.NomCAS,'E1') &&...
            isfield(Temp.NomCAS,'E2') &&...
            isfield(Temp.NomCAS,'E3')
        NomCAS = Temp.NomCAS;
        disp(['=> "CAS"-variables from ',Name.StlFile_Name,...
            '_4_NomCAS.mat has been loaded']);
        Plot(Temp.NomCAS,PlotFlag)
        return
    end
end
% Prepare local values & Functions
%-------------------------------------------------------------------------
CLineDer1  = fnder(CLineFun,1);
CLineDer2  = fnder(CLineFun,2);
%% Choose the interpolated points
%-------------------------------------------------------------------------
CLinLength   =
LengthOfcscvn(CLineFun,CLineFun.breaks(1),CLineFun.breaks(end),...
                           Param.LengthOfcscvnTol);
NrOfSegments = ceil(CLinLength/Param.SliceDist);
LengthAlongCline = linspace(0,CLinLength,NrOfSegments);
TT = TimeOfcscvn(CLineFun,CLineFun.breaks(1),CLineFun.breaks(end),...
        LengthAlongCline,Param.TimeOfcscvnTol);
NomCAS.TimeAlongCline   = TT;
NomCAS.LengthAlongCline = LengthAlongCline;
% Initialisation
%-------------------------------------------------------------------------
CASNrOf   = size(TT,2);
Origin    = NaN*ones(3,CASNrOf);
E1        = NaN*ones(3,CASNrOf);
E2        = NaN*ones(3,CASNrOf);
E3        = NaN*ones(3,CASNrOf);
Ok = 1;
% Find the First CAS with Frenet
E1_raw = ppval(CLineDer1,TT(1));
E1(:,1) = E1_raw/norm(E1_raw);
E2_raw = ppval(CLineDer2,TT(1))-dot(ppval(CLineDer2,TT(1)),...
    E1(:,1))*E1(:,1);
E2(:,1) = E2_raw/norm(E2_raw);
```

```
E3_raw = cross(E1(:,1),E2(:,1));
E3(:,1) = E3_raw/norm(E3_raw);
Origin(:,1) = ppval(CLineFun,TT(1));
%% Loop to find the other CAS
%--------------------------------------------------------------------------
for counter = 2:size(TT,2)
    CLTime = TT(counter);
    % Calculate the Origin
    Origin(:,counter) = ppval(CLineFun,CLTime);
    % Calculate the E1, E2 & E3 Vectors
    E1(:,counter) =
ppval(CLineDer1,CLTime)/norm(ppval(CLineDer1,CLTime));
    % Register the axis
    % (see Normal orientation methods for 3D offset curves,sweep
    %  surfaces and skinning, Computer Graphics Forum, Volume: vol.11,
    %  no.3, Pages: C449-57, Published: 1992)
    E2raw = E2(:,counter-1) - ...
        dot(E2(:,counter-1),E1(:,counter-1))*E1(:,counter-1);
    E2(:,counter) = E2raw/norm(E2raw);
    E3raw = cross(E1(:,counter),E2(:,counter));
    E3(:,counter) = E3raw/norm(E3raw);
end
NomCAS.Origin = Origin';
NomCAS.E1     = E1';
NomCAS.E2     = E2';
NomCAS.E3     = E3';
save([Name.StlFile_Name,'_4_NomCAS.mat'],'NomCAS')
Plot(NomCAS,PlotFlag)
end
function Plot(NomCAS,PlotFlag)
    global F;
    global V;
    global Param
    if PlotFlag == 1
        H.ClineFig = figure('Position',Param.FigPos,...
            'NumberTitle','off',...
            'Name','GetNominalCAS: Nominal CAS');
        SurfH      = trisurf(F,V(:,1),V(:,2),V(:,3),0,'FaceAlpha',0.1,...
                             'EdgeAlpha',0.1);
        hold on;
        plot3(NomCAS.Origin(:,1),NomCAS.Origin(:,2),NomCAS.Origin(:,3),'-
b')
        for i = 1:size(NomCAS.Origin,1)
quiver3(NomCAS.Origin(i,1),NomCAS.Origin(i,2),NomCAS.Origin(i,3),...
                NomCAS.E1(i,1),NomCAS.E1(i,2),NomCAS.E1(i,3),'r');
quiver3(NomCAS.Origin(i,1),NomCAS.Origin(i,2),NomCAS.Origin(i,3),...
                NomCAS.E2(i,1),NomCAS.E2(i,2),NomCAS.E2(i,3),'g');
quiver3(NomCAS.Origin(i,1),NomCAS.Origin(i,2),NomCAS.Origin(i,3),...
                NomCAS.E3(i,1),NomCAS.E3(i,2),NomCAS.E3(i,3),'b');
        end
        axis equal; axis vis3d;
        drawnow
    end
end
```

GetPointsOnVWall.m

```
function GetPointsOnVWall(FaceFlag,PlotFlag)
display('GetPointsOnVWall')
%% Global Values
%--------------------------------------------------------------------------
global F; global V;
global NomCAS
global Name
global VessPts
%% See if the variables exists
%--------------------------------------------------------------------------
if exist([Name.StlFile_Name,'_5_VessPts.mat'],'file')
    Temp = load([Name.StlFile_Name,'_5_VessPts.mat'],'VessPts');
    if isfield(Temp,'VessPts')
        VessPts = Temp.VessPts;
        Plot(Temp.VessPts,PlotFlag)
        disp(['=> "VessPts" from ',Name.StlFile_Name,...
            '_5_VessPts.mat have been loaded'])
        return
    end
end
%% Functions
%--------------------------------------------------------------------------
DPiPt = @(E1,Point,x0) dot(E1, (x0-Point));
%% Find the points of interesection
%--------------------------------------------------------------------------
CoM = NomCAS.Origin;
E1  = NomCAS.E1;
%Take all the triangles except the covers, change the variable for
%NewFlag?
AllTriButFaceFlag = (FaceFlag==0);
VessPts(1).Coord = [];
for i= 2:size(CoM,1)-1
    AllPointInt = [];
    for Tri=1:length(AllTriButFaceFlag)
        if AllTriButFaceFlag(Tri)==1
            j = Tri;
            % distance point-plane:
            dist1 = DPiPt(E1(i,:),CoM(i,:),V(F(j,1),:));
            dist2 = DPiPt(E1(i,:),CoM(i,:),V(F(j,2),:));
            dist3 = DPiPt(E1(i,:),CoM(i,:),V(F(j,3),:));
            testbig   = [dist1,dist2,dist3]  > 0;
            testsmall = [dist1,dist2,dist3]  < 0;
            if sum(testbig)>0 && sum(testsmall)>0
                [PointInt1,check1] =
plane_line_intersect(E1(i,:),CoM(i,:),V(F(j,1),:),V(F(j,2),:));
                [PointInt2,check2] =
plane_line_intersect(E1(i,:),CoM(i,:),V(F(j,1),:),V(F(j,3),:));
                 if check1 == 1 && check2 == 1
                     AllPointInt = [AllPointInt; PointInt1; PointInt2];
                 elseif check1 == 1
                     [PointInt3,check3] =
plane_line_intersect(E1(i,:),CoM(i,:),V(F(j,2),:),V(F(j,3),:));
                     AllPointInt = [AllPointInt; PointInt1; PointInt3];
```

```
                elseif check2 == 1
                    [PointInt3,check3] =
plane_line_intersect(E1(i,:),CoM(i,:),V(F(j,2),:),V(F(j,3),:));
                    AllPointInt = [AllPointInt; PointInt2; PointInt3];
                end
            end
        end
    end
    VessPts(i).Coord = unique(AllPointInt,'rows');
end
%% Locate the Face points
%--------------------------------------------------------------------------
for i=1:length(unique(FaceFlag))-1
    FacePtsIdMtxForm = F(find(FaceFlag==i),:);
    FacePtsIdRowForm = unique(reshape(FacePtsIdMtxForm,1,[]));
    FacePts          = V(FacePtsIdRowForm(:),:);
    WallPtsIdMtxForm = F(find(FaceFlag==0),:);
    WallPtsIdRawForm = unique(reshape(WallPtsIdMtxForm,1,[]));
    WallPts          = V(WallPtsIdRawForm(:),:);
    IntsctFaceFlag = ismember(FacePts,WallPts,'rows');
    IntsctPtsId    = FacePtsIdRowForm(find(IntsctFaceFlag~=0));
    LimitPts = V(IntsctPtsId,:);
    if i==1
        VessPts(1).Coord = unique(LimitPts,'rows');
    else
        VessPts(size(CoM,1)).Coord = unique(LimitPts,'rows');
    end
end
save([Name.StlFile_Name,'_5_VessPts.mat'],'VessPts')
Plot(VessPts,PlotFlag)
end
function Plot(VessPts,PlotFlag)
global V;
global F;
global Param;
% VessPts(1,1).Coord(1,:)
if PlotFlag == 1
    % Initiate the figure
    figure('Position',Param.FigPos,'Name','GetPointsOnVWall',...
        'NumberTitle','off')
    trisurf(F,V(:,1),V(:,2),V(:,3),0,'FaceAlpha',0.1,'EdgeAlpha',0.1);
    hold on;
    axis equal; axis vis3d;
    for i=1:size(VessPts,2)
        % Plot the points
        plot3(VessPts(i).Coord(:,1),...
              VessPts(i).Coord(:,2),...
              VessPts(i).Coord(:,3),'*m');
    end
end
end
```

GetRadInterpolation.m

```
function GoalRValues = GetRadInterpolation(RadFun,PlotFlag)
display('GetRadInterpolation...')
%% Global Values
%--------------------------------------------------------------------------
global NomCAS
global Param
global Stent
%% Find the interpolated Radius (with linear interpolation):
%##########################################################################
OldRValues = ppval(RadFun, NomCAS.LengthAlongCline);
if not(isempty(Param.OptimSlices))
    RValBefore = ppval(RadFun,
NomCAS.LengthAlongCline(Param.OptimSlices(1)));
    RValAfter  = ppval(RadFun,
NomCAS.LengthAlongCline(Param.OptimSlices(end)));
    NewRadBefore = ppval(RadFun,
NomCAS.LengthAlongCline(1:Param.OptimSlices(1)));
    NewRadinterp =
interp1([NomCAS.LengthAlongCline(Param.OptimSlices(1)),...
NomCAS.LengthAlongCline(Param.OptimSlices(end))],...
                            [RValBefore,RValAfter],...
NomCAS.LengthAlongCline(Param.OptimSlices(1)+1:...
Param.OptimSlices(end)-1),...
                            'linear');
    NewRadAfter  = ppval(RadFun,
NomCAS.LengthAlongCline(Param.OptimSlices(end):end));
    GoalRValues  = [NewRadBefore,NewRadinterp,NewRadAfter];
else
    GoalRValues  = OldRValues;
end
GoalRValues(1)   = Stent.R0;
GoalRValues(end) = Stent.R0;
%##########################################################################
% %% Find the interpolated Radius (non linear interpolation):
%##########################################################################
% OldRValues = ppval(RadFun, NomCAS.LengthAlongCline);
% % if not(Param.OptimSlices(1)==0) && not(Param.OptimSlices(end)==0)
% if not(isempty(Param.OptimSlices))
%     RValBefore = ppval(RadFun,
NomCAS.LengthAlongCline(1:Param.OptimSlices(1)-1));
%     RValAfter  = ppval(RadFun,
NomCAS.LengthAlongCline(Param.OptimSlices(end)+1:end));
%     DBefore    = NomCAS.LengthAlongCline(1:Param.OptimSlices(1)-1);
%     DAfter     = NomCAS.LengthAlongCline(Param.OptimSlices(end)+1:end);
%     NewRadBefore = ppval(RadFun,
NomCAS.LengthAlongCline(1:Param.OptimSlices(1)-1));
%     NewRadinterp = interp1([DBefore,DAfter],[RValBefore,RValAfter],...
%                             NomCAS.LengthAlongCline(...
Param.OptimSlices(1):Param.OptimSlices(end)),...
%                             'spline');
%     NewRadAfter  = ppval(RadFun, NomCAS.LengthAlongCline(...
%         Param.OptimSlices(end)+1:end));
%     GoalRValues  = [NewRadBefore,NewRadinterp,NewRadAfter];
% else
```

```
%     GoalRValues  = OldRValues;
% end
% GoalRValues(1)   = Stent.R0;
% GoalRValues(end) = Stent.R0;
%#########################################################################
if PlotFlag==1
    figure('Position',Param.FigPos,...
        'Name','GetRadInterpolation: Radius Interpolation',...
        'NumberTitle','off');
    plot(NomCAS.LengthAlongCline,OldRValues,'*-r')
    hold on
    plot(NomCAS.LengthAlongCline,GoalRValues,'o-b')
    legend({'Old Radius Values','Goal Radius Values'},...
        'Location','NorthEastOutside')
    xlim([NomCAS.LengthAlongCline(1),NomCAS.LengthAlongCline(end)]);
end
end
```

GetStentCAS.m

```
function GetStentCAS(CLineFun,RadFun,FigTitle,PlotFlag)
disp('GetStentCAS...')
%% Global Values
%-------------------------------------------------------------------------
global Stent
global Param
global FinalCAS
global Name
%% See if the variables exists
%-------------------------------------------------------------------------
if exist([Name.StlFile_Name,'_7_FinalCAS.mat'],'file')
    Temp = load([Name.StlFile_Name,'_7_FinalCAS.mat'],'FinalCAS');
    if isfield(Temp.FinalCAS,'Origin') &&...
            isfield(Temp.FinalCAS,'E1') &&...
            isfield(Temp.FinalCAS,'E2') &&...
            isfield(Temp.FinalCAS,'E3') &&...
            isfield(Temp.FinalCAS,'CumulDist')
        FinalCAS = Temp.FinalCAS;
        Plot(FinalCAS,FigTitle,PlotFlag)
        disp(['=> "Min/Max Radius variables" from ',Name.StlFile_Name,...
            '_7_FinalCAS.mat have been loaded'])
        return
    end
end
% Prepare local values & Functions
%-------------------------------------------------------------------------
LofCL =
LengthOfcscvn(CLineFun,CLineFun.breaks(1),CLineFun.breaks(end),...
    Param.LengthOfcscvnTol);
CLineDer1  = fnder(CLineFun,1);
CLineDer2  = fnder(CLineFun,2);
DToNextCAS =@(R1) sqrt(Stent.PPitch0^2+Stent.R0^2-R1^2)*...
                      pi/(Stent.CoilNrOf*12);
% Place 12 CAS between the Xing (because if we later want to place a
% different number of segment in between, we can choose between 1,2,3
% or 4 segements, since all of them are divisors of 12)
% N=1 x--|--|--|--|--|--|--|--|--|--|--|--x
% N=2 x--|--|--|--|--|--x--|--|--|--|--|--x
% N=3 x--|--|--|--x--|--|--|--x--|--|--|--x
% N=4 x--|--|--x--|--|--x--|--|--x--|--|--x
% Initialisation
%-------------------------------------------------------------------------
CASNrOf    = floor(Stent.NOfTurns*2*Stent.CoilNrOf*12+1);
Origin     = NaN*ones(3,CASNrOf);
E1         = NaN*ones(3,CASNrOf);
E2         = NaN*ones(3,CASNrOf);
E3         = NaN*ones(3,CASNrOf);
CumulDist = NaN*ones(1,CASNrOf);
Ok = 1;
% Find the first point and set the first CAS
%-------------------------------------------------------------------------
CLDist  = Stent.StartDist;
CLTime  =
TimeOfcscvn(CLineFun,CLineFun.breaks(1),CLineFun.breaks(end),...
                     CLDist,Param.TimeOfcscvnTol);
```

```
Origin(:,1) = ppval(CLineFun,CLTime);
% Find the First CAS with Frenet
E1_raw = ppval(CLineDer1,CLTime);
E1(:,1) = E1_raw/norm(E1_raw);
E2_raw = ppval(CLineDer2,CLTime)-dot(ppval(CLineDer2,CLTime),...
    E1(:,1))*E1(:,1);
E2(:,1) = E2_raw/norm(E2_raw);
E3_raw = cross(E1(:,1),E2(:,1));
E3(:,1) = E3_raw/norm(E3_raw);
CumulDist(1) = CLDist;
ErsatzCLDist = NaN;
%% Loop to find the other CAS
%-------------------------------------------------------------------------
for counter = 2:Stent.NOfTurns*2*Stent.CoilNrOf*12+1;
    % Find the next Origin
    % ----------------------------------------
    % DefRad1:  radius at the beginning of the segment
    % DefRad2:  radius at the end of the segment
    DefRad1      = ppval(RadFun,CLDist);
    DistToNextO1 = DToNextCAS(DefRad1);
    CLDist2      = CLDist + DistToNextO1;
    DefRad2      = ppval(RadFun,CLDist2);
    DistToNextO2 = DToNextCAS(0.5*(DefRad1+DefRad2));
    CLDist       = CLDist + DistToNextO2;
    CLTime       =
TimeOfcscvn(CLineFun,CLineFun.breaks(1),CLineFun.breaks(end),...
                    CLDist,Param.TimeOfcscvnTol);
    CumulDist(counter) = CLDist;
    if and(CumulDist(counter)<LofCL,...
            isnan(ErsatzCLDist));
        % Calculate the Origin
        Origin(:,counter) = ppval(CLineFun,CLTime);
        % Calculate the E1, E2 & E3 Vectors
        E1(:,counter) =
ppval(CLineDer1,CLTime)/norm(ppval(CLineDer1,CLTime));
        % Register the axis
        % (see Normal orientation methods for 3D offset curves,sweep
        %  surfaces and skinning, Computer Graphics Forum, Volume:
vol.11,
        %  no.3, Pages: C449-57, Published: 1992)
        E2raw = E2(:,counter-1) - ...
            dot(E2(:,counter-1),E1(:,counter-1))*E1(:,counter-1);
        E2(:,counter) = E2raw/norm(E2raw);
        E3raw = cross(E1(:,counter),E2(:,counter));
        E3(:,counter) = E3raw/norm(E3raw);
    else
        if Ok==1
            SavedCounter = counter;
            ErsatzRad    = DefRad1;
            ErsatzD      = DToNextCAS(ErsatzRad);
            ErsatzCLDist = CLDist;
        end
        ErsatzCLDist       = ErsatzCLDist + ErsatzD;
        CumulDist(counter) = ErsatzCLDist;
        Ok = 0;
        Origin(:,counter) = Origin(:,counter-1)+...
            ErsatzD*E1(:,counter-1);
        E1(:,counter) = E1(:,counter-1);
```

```
        E2(:,counter) = E2(:,counter-1);
        E3(:,counter) = E3(:,counter-1);
    end
end
%% Save the data
%-------------------------------------------------------------------------
FinalCAS.Origin    = Origin;
FinalCAS.E1        = E1;
FinalCAS.E2        = E2;
FinalCAS.E3        = E3;
FinalCAS.CumulDist = CumulDist;
save([Name.StlFile_Name,'_7_FinalCAS.mat'],'FinalCAS');
if CumulDist(end)>LofCL
    disp('WARNING:')
    disp('--------')
    disp('THE VESSEL LENGTH IS SHORTER THAN THE STENT LENGTH')
    disp('--------')
end
end
function Plot(FinalCAS,FigTitle,PlotFlag)
    % Plot
    if PlotFlag == 1;
        global V;
        global F;
        global Param;
        global Stent;
        figure('Position',Param.FigPos,'Name',['GetStentCAS:
',FigTitle],...
            'NumberTitle','off');
trisurf(F,V(:,1),V(:,2),V(:,3),0,'FaceAlpha',0.1,'EdgeAlpha',0.1);
        hold on;
        CASNrOf   = floor(Stent.NOfTurns*2*Stent.CoilNrOf*12+1);
        % Plot the CLine
        plot3(FinalCAS.Origin(1,:),FinalCAS.Origin(2,:),...
            FinalCAS.Origin(3,:),'.r');
        for i=2:12:CASNrOf;
            quiver3(FinalCAS.Origin(1,i),FinalCAS.Origin(2,i),...
                    FinalCAS.Origin(3,i),...
FinalCAS.E1(1,i),FinalCAS.E1(2,i),FinalCAS.E1(3,i),'r')
            quiver3(FinalCAS.Origin(1,i),FinalCAS.Origin(2,i),...
                    FinalCAS.Origin(3,i),...
FinalCAS.E2(1,i),FinalCAS.E2(2,i),FinalCAS.E2(3,i),'g')
            quiver3(FinalCAS.Origin(1,i),FinalCAS.Origin(2,i),...
                    FinalCAS.Origin(3,i),...
FinalCAS.E3(1,i),FinalCAS.E3(2,i),FinalCAS.E3(3,i),'b')
        end
        grid on; axis equal; axis vis3d;
    end
end
```

GlobalRadDiff.m

```
function Output = GlobRadDiff(SlicesForOptim,DPos,GoalRValues,PlotFlag)
Diff = PosDependentRadius(SlicesForOptim,DPos,PlotFlag)-
GoalRValues(1,SlicesForOptim);
Id = find(Diff<0);
Output = sum(-Diff(Id));%+mean(Diff(Id));
if isnan(Output)
    Output = 0;
end
```

InPolyedron.m

```
function in=InPolyedron(p,t,tnorm,qp)
%errors check
if nargin~=4
    error('4 inputs required')
end
[m,n]=size(p);
if n~=3
    error('Wrong points dimension');
end
[m1,n]=size(t);
if n~=3
    error('Wrong t dimension');
end
[m2,n]=size(tnorm);
if n~=3
    error('Wrong tnorm dimension');
end
if m1~=m2
    error('t dismatch tnorm dimensions');
end
[m,n]=size(qp);
if n~=3
    error('Wrong qp dimension');
end
%% internal parameters
k=1;%boxes subdivison factor
%future improvement find anoptimal value depending on t and qp size
%rectangle for box2triangle map
rect(1)=min(p(:,1));
rect(2)=min(p(:,2));
rect(3)=max(p(:,1));
rect(4)=max(p(:,2));
global firsttoll;
global secondtoll;
firsttoll=1e-9;
secondtoll=1e-10;
%get size data
np=size(p,1);
% nt=size(t,1);
nq=size(qp,1);
%Sort counterclockwise
t=SortCounterclockwise(t,p(:,1),p(:,2));
% Box2tMap
[Box2tMap]=GetBox2tMap(p,t,tnorm,rect,k);
in=false(nq,1);
%get box reference
minx=rect(1);%min(p(:,1));
miny=rect(2);%min(p(:,2));
maxx=rect(3);%max(p(:,1));
maxy=rect(4);%max(p(:,2));
A=(maxx-minx)*(maxy-miny);
step=sqrt(A/(np*k));%step quasi square
nx=floor((maxx-minx)/step);
```


```
ny=floor((maxy-miny)/step);
if nx==0%check thin mapping
    px=firsttoll+maxx-minx;
    nx=1;
else
px=(maxx-minx+firsttoll)/nx;%eps per aumentare il passo
end
if ny==0%check thin mapping
    py=firsttoll+(maxy-miny);
    ny=1;
else
    py=(maxy-miny+firsttoll)/ny;
end
%loop trough all query points
n=zeros(2,1);%edge normal
for i=1:nq
    %make temp scalar
    x=qp(i,1); y=qp(i,2); z=qp(i,3);
    %get box coordinates
    idx=ceil((x-rect(1)+secondtoll)/px);
    if idx<1||idx>nx
        continue%points is outside
    end
    idy=ceil((y-rect(2)+secondtoll)/py);
    if idy<1||idy>ny
        continue%points is outside
    end
    id=ny*(idx-1)+idy;
    %get mapped triagnles
   ttemp=Box2tMap{id,1};
   %loop trough all triangles
   mindist=inf;N=1;
   for j=1:length(ttemp)
       idt=ttemp(j);
       p1=t(idt,1);p2=t(idt,2);p3=t(idt,3);
       %run inside triangle test
       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
       % edge1
       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
       n(1)=-p(p2,2)+p(p1,2);n(2)=p(p2,1)-p(p1,1);%normals to triangle
edge
       test=n(1)*x+n(2)*y-n(1)*p(p1,1)-n(2)*p(p1,2);
       %debug
%        close(figure(1));
%        figure(1)
%        hold on
%        plot(p([p2,p1],1),p([p2,p1],2),'r-')
%        plot(qp(i,1),qp(i,2),'g*')
       if test<0%mettendo l'uguale si escludono i punti sulla superficie
           continue;%test failed pints is outside of the triangle
       end
       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
       % edge2
       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
       n(1)=-p(p3,2)+p(p2,2);n(2)=p(p3,1)-p(p2,1);%normals to triangle
edge
       test=n(1)*x+n(2)*y-n(1)*p(p2,1)-n(2)*p(p2,2);
       %debug
```

```
%         figure(1)
%         hold on
%         plot(p([p3,p2],1),p([p3,p2],2),'r-')
%

        if test<0%mettendo l'uguale si escludono i punti sulla superficie
            continue;%test failed pints is outside of the triangle

        end
         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % edge3
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         n(1)=-p(p1,2)+p(p3,2);n(2)=p(p1,1)-p(p3,1);%normals to triangle
edge
        test=n(1)*x+n(2)*y-n(1)*p(p1,1)-n(2)*p(p1,2);
        %debug
%         figure(1)
%         hold on
%         plot(p([p1,p3],1),p([p1,p3],2),'r-')
        if test<0%mettendo l'uguale si escludono i punti sulla superficie
            continue;%test failed pints is outside of the triangle
        end
        %debug
%         close(figure(2));
%         figure(2);
%         hold on
%         axis equal
%         trisurf(t(idt,:),p(:,1),p(:,2),p(:,3),'facecolor','c');
%         cc=(p(t(idt,1),:)+p(t(idt,2),:)+p(t(idt,3),:))/3;
%        quiver3(cc(1),cc(2),cc(3),tnorm(idt,1),tnorm(idt,2),tnorm(idt,3))
%         plot3(x,y,z,'g*');
%
%     end
    %run semispace test with closest triangle
        n1=tnorm(idt,1);n2=tnorm(idt,2);n3=tnorm(idt,3);
        p1=t(idt,1);
        d=-n1*p(p1,1)-n2*p(p1,2)-n3*p(p1,3);
        %run distance test triangle test
dist=z-(-n1*x-n2*y-d)/n3;%distance along ray
        if abs(dist)<abs(mindist)
            mindist=dist;
            N=n3;
        end
    end
in(i)=mindist*N<=0;
end
end
%% Build2DBoxes
function [Box2tMap]=GetBox2tMap(p,t,tnorm,rect,k)
%rect contiene gli estrameli del rettagnolo di divisione.
%k Ë il fattore di divisione delle boxes, per k=1 il numero di scatole
sar‡
%uguale al numero di punti. k=2 qudruplicher‡ il numero delle scatole
global firsttoll;
global secondtoll;
%size parameters
np=size(p,1);
nt=size(t,1);
```

```
%bounding box analysis
minx=rect(1);%min(p(:,1));
miny=rect(2);%min(p(:,2));
maxx=rect(3);%max(p(:,1));
maxy=rect(4);%max(p(:,2));
A=(maxx-minx)*(maxy-miny);
step=sqrt(A/(np*k));%step quasi square
nx=floor((maxx-minx)/step);%nota: non cambiare floor con ceil
ny=floor((maxy-miny)/step);
if nx==0%check thin mapping
    px=firsttoll+maxx-minx;
    nx=1;
else
px=(maxx-minx+firsttoll)/nx;%eps per aumentare il passo
end
if ny==0%check thin mapping
    py=firsttoll+(maxy-miny);
    ny=1;
else
    py=(maxy-miny+firsttoll)/ny;
end
%nota 1e-9 per aumentare il passo in modo da evitare nel passo successivo
u
%indice pi˘ grande del numero delle scatole
N=nx*ny;%real Boxes number
Box2tMap=cell(N,1);
BoxesId=zeros(np,2,'int32');
%first loop to count points and get ceiling round
for i=1:np
    idx=ceil((p(i,1)-minx+secondtoll)/px);
    idy=ceil((p(i,2)-miny+secondtoll)/py);
%     id=ny*(idx-1)+idy;
     BoxesId(i,1)=idx;
     BoxesId(i,2)=idy;
end
c=zeros(N,1,'int32');
%now loop trough all triangles a build the map
for i=1:nt
    if abs(tnorm(i,3))<0.00000000001%jump vertical triangles
        continue
    end
    p1=t(i,1); p2=t(i,2); p3=t(i,3);
    %get extreme values
    idxmax=max(BoxesId([p1,p2,p3],1));
    idxmin=min(BoxesId([p1,p2,p3],1));
    idymax=max(BoxesId([p1,p2,p3],2));
    idymin=min(BoxesId([p1,p2,p3],2));
    %loop trough all boxes that may contains the triangle
    for idx=idxmin:idxmax
        for idy=idymin:idymax
            %get boxes id and increase counter
            id=ny*(idx-1)+idy;
            c(id)=c(id)+1;
            Box2tMap{id,1}(c(id))=i;%insert traingle into map
        end
    end
```

```
end
end
%% SortCounterclockwise
function t=SortCounterclockwise(t,x,y)
%get points coordinate vectors
x1=x(t(:,1));x2=x(t(:,2));x3=x(t(:,3));
y1=y(t(:,1));y2=y(t(:,2));y3=y(t(:,3));
cx=(x1+x2+x3)/3;cy=(y1+y2+y3)/3;%centroid
clear x3 y3
v1x=x1-cx;v1y=y1-cy;
v2x=x2-cx;v2y=y2-cy;
cp=(v1x.*v2y-v1y.*v2x)<0;%fails cross product criterion
t(cp,:)=t(cp,[2 1 3]);%get counterclockwise orientation
end
```

LengthOfcscvn.m

```
function Int = LengthOfcscvn(Fun,t0,t1list,tol)
% Function
L =@(NewPts) sum(sqrt(sum((NewPts(:,2:end)-NewPts(:,1:end-1)).^2,1)));
Int = NaN*ones(1,length(t1list));
for i = 1:length(t1list)
    t1 = t1list(i);
    % First Step
    OldPts = ppval(Fun,linspace(t0,t1,2));
    NewPts = ppval(Fun,linspace(t0,t1,4));
    OldL   = L(OldPts);
    NewL   = L(NewPts);
    N = 4;
    while abs(OldL-NewL)>tol
        N      = N*2;
        OldL   = NewL;
        NewPts = ppval(Fun,linspace(t0,t1,N));
        NewL   = L(NewPts);
    end
    Int(i) = NewL;
end
```

plane_line_intersect.m

```
function [I,check]=plane_line_intersect(n,V0,P0,P1)
%plane_line_intersect computes the intersection of a plane and a
segment(or
%a straight line)
% Inputs:
%       n: normal vector of the Plane
%       V0: any point that belongs to the Plane
%       P0: end point 1 of the segment P0P1
%       P1:  end point 2 of the segment P0P1
%
%Outputs:
%      I    is the point of interection
%     Check is an indicator:
%      0 => disjoint (no intersection)
%      1 => the plane intersects P0P1 in the unique point I
%      2 => the segment lies in the plane
%      3=>the intersection lies outside the segment P0P1
I=[0 0 0];
u = P1-P0;
w = P0 - V0;
D = dot(n,u);
N = -dot(n,w);
check=0;
if abs(D) < 10^-7        % The segment is parallel to plane
        if N == 0           % The segment lies in plane
            check=2;
            return
        else
            check=0;       %no intersection
            return
        end
end
%compute the intersection parameter
sI = N / D;
I = P0+ sI.*u;
if (sI < 0 || sI > 1)
    check= 3;          %The intersection point  lies outside the segment,
so there is no intersection
else
    check=1;
end
```

PosDependentCurvature.m

```
function Output = PosDependentCurvature(SlicesForOpt,...
    DPos,GoalRValues,PlotFlag)
global NomCAS
% Transform the points
%---------------------
for i = 1:length(SlicesForOpt)
    Slice = SlicesForOpt(1)+i-1;
    Orig     = NomCAS.Origin(Slice,:);
    TrafoMtx = [NomCAS.E1(Slice,:)/norm(NomCAS.E1(Slice,:));...
                NomCAS.E2(Slice,:)/norm(NomCAS.E2(Slice,:));...
                NomCAS.E3(Slice,:)/norm(NomCAS.E3(Slice,:))];
    NewCLinePts(i,:)  = (TrafoMtx\([0;DPos(:,i)])+Orig')';
end
% Plot the transformed points
%----------------------------
if PlotFlag == 1
    figure
    plot3(NomCAS.Origin(:,1),NomCAS.Origin(:,2),NomCAS.Origin(:,3),'.-r')
    hold on
    plot3(NewCLinePts(:,1),NewCLinePts(:,2),NewCLinePts(:,3),'.b')
end
% Gather the points and build the partial function
%-------------------------------------------------
ExternalPtsNrOf = 3;
SlicesBefore    = SlicesForOpt(1)-ExternalPtsNrOf:SlicesForOpt(1)-1;
SlicesAfter     = SlicesForOpt(end)+1:SlicesForOpt(end)+ExternalPtsNrOf;
x1 =
[NomCAS.Origin(SlicesBefore,1);NewCLinePts(:,1);NomCAS.Origin(SlicesAfter
,1)];
y1 =
[NomCAS.Origin(SlicesBefore,2);NewCLinePts(:,2);NomCAS.Origin(SlicesAfter
,2)];
z1 =
[NomCAS.Origin(SlicesBefore,3);NewCLinePts(:,3);NomCAS.Origin(SlicesAfter
,3)];
Fun = cscvn([x1,y1,z1]');
% Increase the number of sampling points with the function
%---------------------------------------------------------
NN     = 4;
PtsNr  = NN*length(x1);
Pts = ppval(Fun,linspace(Fun.breaks(1),Fun.breaks(end),PtsNr));
% Build the gradients
%--------------------
dx  = gradient(Pts(1,:));
ddx = gradient(dx);
dy  = gradient(Pts(2,:));
ddy = gradient(dy);
dz  = gradient(Pts(3,:));
ddz = gradient(dz);
% Build the curvature
%--------------------
k = sqrt((ddz.*dy-ddy.*dz).^2+...
         (ddx.*dz-ddz.*dx).^2+...
         (ddy.*dx-ddx.*dy).^2)./...
```

```
        ((dx.^2+dy.^2+dz.^2).^(3/2));

CurvRad = k.^-1;
% Calculate the output
%---------------------
InvKDiff = k(1:NN:end)-
(1*GoalRValues([SlicesBefore,SlicesForOpt,SlicesAfter])).^-1;
Id   = find(InvKDiff>0);
Output = sum(InvKDiff(Id));
end
```

PosDependentRadius.m

```
function GlobalRmin = PosDependentRadius(SliceNrInput,DPosInput,PlotFlag)
% % To Run the function as a script
% %xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
% clear all; clc; close all;
% load('meshstentaneurysmultversio_4_Slices.mat')
% DPosInput = [-1,-1.5;-1,-1.5]';
% SliceNrInput = [82,83];
% PlotFlag = 1;
% %xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
global CVRFun
for j = 1:size(SliceNrInput,2)
    SliceNr = SliceNrInput(j);
    % Load the points
    %----------------
    VessPts = CVRFun.VessPts2D(SliceNr).Coord;
    GlobalRmin(j) = Inf;
    PtIdOfRmin    = 0;
    for i = 1:size(VessPts,2)
        Ract = norm(VessPts(:,i)-DPosInput(:,j));
        if Ract < GlobalRmin(j)
            GlobalRmin(j) = Ract;
            PtIdOfRmin = i;
        end
    end
    if PlotFlag == 1
        figure
        plot(VessPts(1,:),VessPts(2,:),'.r')
        hold on
        plot([DPosInput(1,j),VessPts(1,PtIdOfRmin)],...
             [DPosInput(2,j),VessPts(2,PtIdOfRmin)],'.-b')
        grid on; axis equal;
    end
end
```

ProduceNotOptRadFun.m

```
function RadFun = ProduceNotOptRadFun(Rc,PlotFlag)
disp('ProduceNotOptRadFun...')
global NomCAS
global CVRFun
global Stent
global Param
%% Gather the necessary data for the restrictions
%--------------------------------------------------------------------------
Times = NomCAS.TimeAlongCline;
Rad =
min(min(CVRFun.VessMinRad,Stent.R0*ones(size(CVRFun.VessMinRad))),Rc);
% Change the values around the neck
Rad(Param.OptimSlices) = CVRFun.VessMinRad(Param.OptimSlices);
RadFun = spline(NomCAS.LengthAlongCline,Rad-Param.ROffset);
```

ProduceRadFun.m

```
function [RadFun,Rc] = ProduceRadFun(PlotFlag)
disp('ProduceRadFun...')
IncludeCurvatureRestriction = 1;
%% Global Values
%-------------------------------------------------------------------------
global NomCAS
global Param
global Stent
global CVRFun
% %% Looks if a Flag File exists
% %-------------------------------------------------------------------------
% if exist([Name.StlFile_Name,'_7_RadFun.mat'])
%     load([Name.StlFile_Name,'_7_RadFun.mat'],'RadFun')
%     load([Name.StlFile_Name,'_7_RadFunRc.mat'],'Rc')
%     disp(['=> ',Name.StlFile_Name,'_7_RadFun.mat',' has been loaded'])
%     Plot(RadFun,Rc,PlotFlag)
%     return
% end
%% Gather the necessary data for the restrictions
%-------------------------------------------------------------------------
Times = NomCAS.TimeAlongCline;
Rc    = GetInitCurvature(NomCAS);
if IncludeCurvatureRestriction == 1
    Rad   =
min(min(CVRFun.VessMinRad,Stent.R0*ones(size(CVRFun.VessMinRad))),Rc);
else
    Rad   =
min(CVRFun.VessMinRad,Stent.R0*ones(size(CVRFun.VessMinRad)));
end
%% Produce the Midline function
%-------------------------------------------------------------------------
RadFun = spline(NomCAS.LengthAlongCline,Rad-Param.ROffset);
% save([Name.StlFile_Name,'_7_RadFun.mat'],'RadFun')
% save([Name.StlFile_Name,'_7_RadFunRc.mat'],'Rc')
if IncludeCurvatureRestriction == 1
    Plot(RadFun,PlotFlag,Rc)
else
    Plot(RadFun,PlotFlag)
end
end
function Plot(RadFun,PlotFlag,varargin)
Rc = varargin{1};
%% Plot
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
if PlotFlag ==1;
    global Param
    global NomCAS
    global Stent
    global CVRFun
    figure('Name','ProduceRadFun: Radius Function',...
        'NumberTitle','off','Position',Param.FigPos);
    % Plot the Vessel limitation
    plot(NomCAS.LengthAlongCline,CVRFun.VessMinRad,'.-r'); hold on;
    % Plot the nominal Stend Radius limitation
```

```
plot(NomCAS.LengthAlongCline,Stent.R0*ones(size(CVRFun.VessMinRad)),'.-
b');
    if not(isempty(varargin))
        plot(NomCAS.LengthAlongCline,Rc,'.-m');
    end
    % Plot the spline
    Ill = linspace(RadFun.breaks(1),RadFun.breaks(end),...
                   10*length(RadFun.breaks));
    RRl  = ppval(RadFun,Ill);
    plot(Ill,RRl,'-k')
    xlabel('Length Along Curve [mm]'); ylabel('Stent Radius [mm]'); grid
on
    xlim([min(NomCAS.LengthAlongCline),max(NomCAS.LengthAlongCline)])
    if not(isempty(varargin))
        legend({'Vessel Radius','Nominal Stent Radius',...
        'Curvature Restricted Radius','Interpolation Function'})
    else
        legend({'Vessel Radius','Nominal Stent Radius',...
        'Interpolation Function'})
    end
    ylim([0,1.5*Stent.R0])
end
end
```

ProduceRadFunNew.m

```
function RadFunSmooth = ProduceRadFunNew(NewRad,Rc,GoalRValues,PlotFlag)
disp('ProduceRadFunNew...')
%% Global Values
%--------------------------------------------------------------------------
global Slices
global Name
global Stent
global Param
global NomCAS
global OptCLineFunL
%% Looks if a Flag File exists
%--------------------------------------------------------------------------
if exist([Name.StlFile_Name,'_5d_RadFunNew.mat'])
    load([Name.StlFile_Name,'_5d_RadFunNew.mat'],'RadFun')
    disp(['=> ',Name.StlFile_Name,'_5d_RadFunNew.mat',' has been
loaded'])
    Plot(RadFun,NewRad,Rc,RadFunSmooth,PlotFlag)
    return
end
%% Gather the necessary data for the restrictions
%--------------------------------------------------------------------------
% size(NewRad)
% size(Stent.R0*ones(1,size(NomCAS.TimeAlongCline,2)))
% size(Rc)
Rad   = min(...
         min(...
          min(...
           NewRad,Stent.R0*ones(1,size(NomCAS.TimeAlongCline,2))),...
            Rc),...
             GoalRValues);
% Smoothing the Function
RadSmooth = smooth(OptCLineFunL,Rad,0.2,'lowess');
RadSmooth = RadSmooth-Param.ROffset;
%% Produce the Midline function
%--------------------------------------------------------------------------
RadFun       = spline(OptCLineFunL,Rad);
RadFunSmooth = spline(OptCLineFunL,RadSmooth);
% save([Name.StlFile_Name,'_5d_RadFunNew.mat'],'RadFun')
Plot(RadFun,NewRad,Rc,GoalRValues,RadFunSmooth,PlotFlag)
end
function Plot(RadFun,NewRad,Rc,GoalRValues,RadFunSmooth,PlotFlag)
%% Plot
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
if PlotFlag ==1;
    global Param
    global Slices
    global Stent
    global OptCLineFunL
    figure('Position',Param.FigPos,...
        'Name','ProduceRadFunNew: Radius Function',...
        'NumberTitle','off');
    % Plot the Vessel limitation
    plot(OptCLineFunL,NewRad,'.-r'); hold on;
```

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

```
    % Plot the nominal Stend Radius limitation
    plot(OptCLineFunL,Stent.R0*ones(size(OptCLineFunL)),'.-b');
    % Plot the restriction due to the curvature
    plot(OptCLineFunL,Rc,'.-m')
    % Plot the restriction due to the goal value
    plot(OptCLineFunL,GoalRValues,'.-g')
    % Plot the smoothed-offset radius function
    plot(OptCLineFunL,...
        ppval(RadFunSmooth,RadFunSmooth.breaks),'.-k')
    xlabel('Length Along Curve [mm]'); ylabel('Stent Radius [mm]'); grid
on
    xlim([min(OptCLineFunL),max(OptCLineFunL)])
    ylim([0,1.2*Stent.R0])
    legend({'Vessel Radius','Nominal Stent Radius',...
        'Curvature Restricted Radius','GoalRValues',...
        'Smoothed Offsetted Function'})
end
end
```

SelectMeshTri.m

```
function [TriFlags,WallPts] = SelectMeshTri(varargin)
disp('SelectMeshTri...')
%% Global Values
%-------------------------------------------------------------------------
global F; global V;
%% INPUTS:
%  -------
% varargin : empty or handle to the figure
if isempty(varargin)
    % plot the surface
    Scrsz = get(0,'Screensize');
    FigPos = [100,100,Scrsz(3)-200,Scrsz(4)-200];
    FigH = figure('Position',FigPos);
    trisurf(F,V(:,1),V(:,2),V(:,3),1);
    axis equal; grid on; box on;
else
    FigH = varargin{1};
end
% Get Min and max values
minX = min(V(:,1));  maxX = max(V(:,1));
minY = min(V(:,2));  maxY = max(V(:,2));
minZ = min(V(:,3));  maxZ = max(V(:,3));
XRange = maxX-minX;
YRange = maxY-minY;
ZRange = maxZ-minZ;
% Prepare the line Handle
hold on
LineH = plot3(NaN,NaN,NaN,'.-r','LineWidth',2);
% Extent the axis
factor = 0.2;
set(gca,'XLim',[minX-factor*XRange,maxX+factor*XRange],...
        'YLim',[minY-factor*YRange,maxY+factor*YRange],...
        'ZLim',[minZ-factor*ZRange,maxZ+factor*ZRange])
% Create a callback function
set(FigH,'windowbuttondownfcn',@SaveMousePosition);
% Create a structure containing the handle to the object with a tag
data = guihandles(FigH);
% Include other parameters to the data structure
data.Positions = [];
data.LineH     = LineH;
% Save the change to the gui data
guidata(FigH,data);
% Select the points
uiwait(FigH); % + Select the points for CM
% Do the points selection
%------------------------
% Prepare the x0 points ( the centroids of each triangle)
x0ik = zeros(size(F,1),3);
for i=1:size(F,1)
    x0ik(i,1:3) = mean([V(F(i,:),1),V(F(i,:),2),V(F(i,:),3)]);
end
% Flag the triangles that are betwenn the walls
TriFlags = FlagPtsBetwWalls(x0ik,data.Positions,1);
WallPts  = data.Positions;
```

```
delete(LineH)
%% CallBack Function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  SaveMousePosition(hObject,event)
% Load the guidata
data = guidata(FigH);
% Check if left or right click
Button = get(FigH,'SelectionType');
if strcmp(Button,'normal')
    % Get the actual position
    ActPos = get(gca,'CurrentPoint');
    FrontPts = ActPos(1,:);
    BackPts  = ActPos(2,:);
    % Update the Positions
    PtNr = size(data.Positions,1)+1;
    data.Positions(PtNr,1,:) = FrontPts;
    data.Positions(PtNr,2,:) = BackPts;
    % Plot the selected points
    hold on
    set(data.LineH,'XData',data.Positions(1:PtNr,1,1),...
                   'YData',data.Positions(1:PtNr,1,2),...
                   'ZData',data.Positions(1:PtNr,1,3));
    % Trick to plot the line in any orientation
    %-----------------------------------------
    Orientation = get(gca,'CameraPosition')==get(gca,'CameraTarget');
    if Orientation(1)==0 && Orientation(2)==1 && Orientation(3)==1
        axis 'auto x'
    elseif Orientation(1)==1 && Orientation(2)==0 && Orientation(3)==1
        axis 'auto y'
    elseif Orientation(1)==1 && Orientation(2)==1 && Orientation(3)==0
        axis 'auto z'
    end
    drawnow
else
    if not(isempty(data.Positions))
        set(data.LineH,'XData',data.Positions([1:end,1],1,1),...
                       'YData',data.Positions([1:end,1],1,2),...
                       'ZData',data.Positions([1:end,1],1,3));
        drawnow
        uiresume(FigH)
    end
end
% Save possible change to the gui data
guidata(FigH,data);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end
```


SelectMeshTriRect.m

```
function [TriFlags,WallPts] = SelectMeshTriRect(F,V,varargin)
disp('SelectMeshTriRect...')
%% INPUTS:
%  -------
% F : Element Mtx
% V : Vertice Mtx
% varargin : empty or handle to the figure
global Param;
if isempty(varargin)
    % plot the surface
    Scrsz = get(0,'Screensize');
    FigH = figure('Position',Param.FigPos);
    trisurf(F,V(:,1),V(:,2),V(:,3),1);
    axis equal; grid on; box on;
else
    FigH = varargin{1};
end
% Get Min and max values
minX = min(V(:,1));  maxX = max(V(:,1));
minY = min(V(:,2));  maxY = max(V(:,2));
minZ = min(V(:,3));  maxZ = max(V(:,3));
XRange = maxX-minX;
YRange = maxY-minY;
ZRange = maxZ-minZ;
% Prepare the line Handle
hold on
LineH = plot3(NaN,NaN,NaN,'.-r','LineWidth',2);
axis vis3d
% Extent the axis
factor = 0.2;
set(gca,'XLim',[minX-factor*XRange,maxX+factor*XRange],...
        'YLim',[minY-factor*YRange,maxY+factor*YRange],...
        'ZLim',[minZ-factor*ZRange,maxZ+factor*ZRange])
% Create a callback function
set(FigH,'windowbuttondownfcn',@SaveMousePosition);
% Create a structure containing the handle to the object with a tag
data = guihandles(FigH);
% Include other parameters to the data structure
data.Pos   = [];
data.LineH = LineH;
% Save the change to the gui data
guidata(FigH,data);
% Select the points
uiwait(FigH); % + Select the points for CM
% Do the points selection
%------------------------
% Prepare the x0 points ( the centroids of each triangle)
x0ik = zeros(size(F,1),3);
for i=1:size(F,1)
    x0ik(i,1:3) = mean([V(F(i,:),1),V(F(i,:),2),V(F(i,:),3)]);
end
% Flag the triangles that are betwenn the walls
TriFlags = FlagPtsBetwWalls(x0ik,data.Pos,1);
WallPts  = data.Pos;
```

```
delete(LineH)
set(FigH,'windowbuttondownfcn',[]);
%% CallBack Function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  SaveMousePosition(hObject,event)
% Load the guidata
data = guidata(FigH);
% Check if left or right click
Button = get(FigH,'SelectionType');
if strcmp(Button,'normal')
    % Get the actual position
    ActPos   = get(gca,'CurrentPoint');
    FrontPt  = ActPos(1,:);
    BackPt   = ActPos(2,:);
    DepthVec = ActPos(2,:)-ActPos(1,:);
    % Update the Positions
    PtNr = size(data.Pos,1)+1;
    % Action depending on the number of selected points
    %   1----- LL -----2
    %   |              |
    %   ll             |
    %   |              |
    %  4*--------------3      * = not selected but calculated
    if PtNr==1
        % Register the position
        data.Pos(PtNr,1,:) = FrontPt+0.05*DepthVec;
        data.Pos(PtNr,2,:) = BackPt;
        % Plot the selected points
        hold on
        set(data.LineH,'XData',data.Pos(1:PtNr,1,1),...
                       'YData',data.Pos(1:PtNr,1,2),...
                       'ZData',data.Pos(1:PtNr,1,3));
    elseif PtNr==2

        % Register the position
        data.Pos(PtNr,1,:) = FrontPt+0.05*DepthVec;
        data.Pos(PtNr,2,:) = BackPt;

        % Plot the selected points
        hold on
        set(data.LineH,'XData',data.Pos(1:PtNr,1,1),...
                       'YData',data.Pos(1:PtNr,1,2),...
                       'ZData',data.Pos(1:PtNr,1,3));
    elseif PtNr==3
        % Register the position
        data.Pos(PtNr,1,:) = FrontPt+0.05*DepthVec;
        data.Pos(PtNr,2,:) = BackPt;
        % Calculate v23
        v23(1,1:3) = data.Pos(3,1,:)-data.Pos(2,1,:);
        % Calculat the 4th point
        FrntPt1(1,1:3) = data.Pos(1,1,1:3);
        BackPt1(1,1:3) = data.Pos(1,2,1:3);
        FrntPt4(1,1:3) = FrntPt1(1,1:3)+v23(1,1:3);
        BackPt4(1,1:3) = BackPt1(1,1:3)+v23(1,1:3);
        data.Pos(PtNr+1,1,1:3) = FrntPt4(1,1:3);
        data.Pos(PtNr+1,2,1:3) = BackPt4(1,1:3);
        % Plot the rectangle
        hold on
```

```
        set(data.LineH,'XData',data.Pos([1:PtNr+1,1],1,1),...
                       'YData',data.Pos([1:PtNr+1,1],1,2),...
                       'ZData',data.Pos([1:PtNr+1,1],1,3));
    else
        % Reset everything to zero ???
        Ans = questdlg('Do you want to start again?','Question',...
                       'Yes','No','No');
        if strcmp(Ans,'Yes')
            % Reset everything to zero
            data.Pos = [];
            set(data.LineH,'XData',NaN,'YData',NaN,'ZData',NaN);
        else
            uiresume(FigH)
        end
    end
else
    set(data.LineH,'XData',data.Pos([1:end,1],1,1),...
                   'YData',data.Pos([1:end,1],1,2),...
                   'ZData',data.Pos([1:end,1],1,3));
    drawnow
    if size(data.Pos,1)>=3
        uiresume(FigH)
    end
end
% Save possible change to the gui data
guidata(FigH,data);
end
end
```

SelectOneTri.m

```
function Output = SelectOneTri(FigH)
disp('SelectOneTri...')
%% Global Values
%--------------------------------------------------------------------------
global F; global V; global N;
% Get Min and max values
minX = min(V(:,1));
maxX = max(V(:,1));
minY = min(V(:,2));
maxY = max(V(:,2));
minZ = min(V(:,3));
maxZ = max(V(:,3));
XRange = maxX-minX;
YRange = maxY-minY;
ZRange = maxZ-minZ;
% Prepare the line Handle
hold on
TriH = plot3(NaN,NaN,NaN,'*-m');
% Extent the axis
factor = 0.2;
set(gca,'XLim',[minX-factor*XRange,maxX+factor*XRange],...
        'YLim',[minY-factor*YRange,maxY+factor*YRange],...
        'ZLim',[minZ-factor*ZRange,maxZ+factor*ZRange])
% Create a callback function
set(FigH,'windowbuttondownfcn',@SaveMousePosition);
% Create a structure containing the handle to the object with a tag
data = guihandles(FigH);
% Include other parameters to the data structure
data.TriH     = TriH;
% Save the change to the gui data
guidata(FigH,data);
% wait till the figure is closed
uiwait(FigH);
Output = PtNr;
delete(TriH)
%% CallBack Function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function SaveMousePosition(hObject,event)
% More info:
% http://www.angelfire.com/fl/houseofbartlett/solutions/line2tri.html
% Load the guidata
data = guidata(FigH);
% Check if left or right click
Button = get(FigH,'SelectionType');
if strcmp(Button,'normal')
    % Get the actual position
    ActPos = get(gca,'CurrentPoint');
    % Plot the ray
%     hold on
%     plot3(ActPos(:,1),ActPos(:,2),ActPos(:,3),'.-b');
    % Do the vector and point of the ray
    Vect = (ActPos(2,:)-ActPos(1,:));
    Pt   = ActPos(1,:);
    Ptt  = Inf;
```

```
    % Search for the closest intersecting triangle
    for i=1:size(F,1)
        PtId = F(i,:);
        Pt1  = V(PtId(1),:);
        Pt2  = V(PtId(2),:);
        Pt3  = V(PtId(3),:);
        % Compute the normal
        Norm(1) = N(i,1);
        Norm(2) = N(i,2);
        Norm(3) = N(i,3);
        % dot product of normal and line's vector if zero line is
parallel to triangle
        dotprod = Norm(1)*Vect(1) + Norm(2)*Vect(2) + Norm(3)*Vect(3);
        if dotprod < 0
            %Find point of intersect to triangle plane.
            %find t to intersect point
            t = -(Norm(1)*(Pt(1)-Pt1(1))+Norm(2)*(Pt(2)-
Pt1(2))+Norm(3)*(Pt(3)-Pt1(3)))/...
                    (Norm(1)*Vect(1)+Norm(2)*Vect(2)+Norm(3)*Vect(3));
            % if ds is neg line started past triangle so can't hit
triangle.
            if t > 0
                IntPt(1) = Pt(1) + Vect(1)*t;
                IntPt(2) = Pt(2) + Vect(2)*t;
                IntPt(3) = Pt(3) + Vect(3)*t;
                if Clockness(Pt1',Pt2',IntPt,Norm)==1
                    if Clockness(Pt2,Pt3,IntPt,Norm)==1
                        if Clockness(Pt3,Pt1,IntPt,Norm)==1
                            if t<Ptt
                                PtNr = i;
                                Ptt  = t;
                            end
                        end
                    end
                end
            end
        end
    end
    PtId = F(PtNr,:);
    hold on
    set(data.TriH,'XData',V(PtId([1:end,1]),1),...
                  'YData',V(PtId([1:end,1]),2),...
                  'ZData',V(PtId([1:end,1]),3))
    drawnow
else
    if exist('PtNr')
        uiresume(FigH)
    end
end
end
% Clockness Function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Out = Clockness(Pt1,Pt2,Pt3,Norm)
   % normal of trinagle
   testi = (((Pt2(2) - Pt1(2))*(Pt3(3) - Pt1(3))) - ((Pt3(2) -
Pt1(2))*(Pt2(3) - Pt1(3))));
   testj = (((Pt2(3) - Pt1(3))*(Pt3(1) - Pt1(1))) - ((Pt3(3) -
Pt1(3))*(Pt2(1) - Pt1(1))));
```

```
   testk = (((Pt2(1) - Pt1(1))*(Pt3(2) - Pt1(2))) - ((Pt3(1) -
Pt1(1))*(Pt2(2) - Pt1(2))));
   % Dot product with triangle normal
   Temp = testi*Norm(1) + testj*Norm(2) + testk*Norm(3);
   % answer
   if Temp < 0
       Out = 0;
   else
       Out = 1;
   end
end
end
```

SmoothMidLine.m

```
function OptCLineFun = SmoothMidLine(NewCLinePts)
disp('SmoothMidLine...')
global OptCLineFunL
global Param
global NomCAS
Method = 1;
if Method == 1
    % Create the smoothed CLine Pts
    %------------------------------
    NewCLinePtsS = NewCLinePts;
    SliceRange = Param.OptimSlices(1)  -Param.NPtsArndSmtg:...
                 Param.OptimSlices(end)+Param.NPtsArndSmtg;
    CLineFun     = cscvn(NewCLinePts');
    OptCLineFunL = LengthOfcscvn(CLineFun,CLineFun.breaks(1),...
                        CLineFun.breaks,Param.LengthOfcscvnTol);
    NewCLinePtsS(SliceRange,1) = smooth(OptCLineFunL(SliceRange),...
NewCLinePts(SliceRange,1),Param.SmtgCoef,'sgolay',Param.SmtgPolModel);
    NewCLinePtsS(SliceRange,2) = smooth(OptCLineFunL(SliceRange),...
NewCLinePts(SliceRange,2),Param.SmtgCoef,'sgolay',Param.SmtgPolModel);
    NewCLinePtsS(SliceRange,3) = smooth(OptCLineFunL(SliceRange),...
NewCLinePts(SliceRange,3),Param.SmtgCoef,'sgolay',Param.SmtgPolModel);
    % Do the "transition phase" to the non-smoothed points
    %-----------------------------------------------------
    NonSmoothedPts = NewCLinePts;
    Weight = linspace(0,1,Param.NPtsArndSmtg+2);
    Weight = Weight(2:end-1);
    % Points before the Neck
    c = 1;
    for i=Param.OptimSlices(1)-Param.NPtsArndSmtg:Param.OptimSlices(1)-1
        NonSmoothedPts(i,:) = ...
            (1-Weight(c))*NewCLinePts(i,:)+...
            Weight(c)*NewCLinePtsS(i,:);
        c = c+1;
    end
    % Points on the Neck
    NonSmoothedPts(Param.OptimSlices,:) =
NewCLinePtsS(Param.OptimSlices,:);
    % Points after the Neck
    c = 1;
    for
i=Param.OptimSlices(end)+1:Param.OptimSlices(end)+Param.NPtsArndSmtg
        NonSmoothedPts(i,:) = ...
            Weight(c)*NewCLinePts(i,:)+...
            (1-Weight(c))*NewCLinePtsS(i,:);
        c = c+1;
    end
    % Do a last smooting
    OptCLineFun1  = cscvn(NonSmoothedPts');
    OptCLineFun2L = LengthOfcscvn(OptCLineFun1,OptCLineFun1.breaks(1),...
                        OptCLineFun1.breaks,Param.LengthOfcscvnTol);
    OptCLineFun1_Pts = ppval(OptCLineFun1,OptCLineFun1.breaks);
    OptCLineFun2_PtsS(:,1) =
smooth(OptCLineFun2L,OptCLineFun1_Pts(1,:),...
        0.025,'sgolay',3);
    OptCLineFun2_PtsS(:,2) =
```

```
smooth(OptCLineFun2L,OptCLineFun1_Pts(2,:),...
        0.025,'sgolay',3);
    OptCLineFun2_PtsS(:,3) =
smooth(OptCLineFun2L,OptCLineFun1_Pts(3,:),...
        0.025,'sgolay',3);
    % Plot
    %--------------------------------------------------------------------
    Plot(NomCAS,OptCLineFun2_PtsS,1)
    % Prepare the Output
    %--------------------------------------------------------------------
    OptCLineFun  = cscvn(OptCLineFun2_PtsS');
else
     % Create the smoothed CLine Pts
    %-----------------------------
    NewCLinePtsS = NewCLinePts;
    SliceRange = Param.OptimSlices(1)  -Param.NPtsArndSmtg:...
                 Param.OptimSlices(end)+Param.NPtsArndSmtg;
    CLineFun     = cscvn(NewCLinePts');
    OptCLineFunL = LengthOfcscvn(CLineFun,CLineFun.breaks(1),...
                        CLineFun.breaks,Param.LengthOfcscvnTol);
    NewCLinePtsS(SliceRange,1) = smooth(OptCLineFunL(SliceRange),...
NewCLinePts(SliceRange,1),Param.SmtgCoef,'sgolay',Param.SmtgPolModel);
    NewCLinePtsS(SliceRange,2) = smooth(OptCLineFunL(SliceRange),...
NewCLinePts(SliceRange,2),Param.SmtgCoef,'sgolay',Param.SmtgPolModel);
    NewCLinePtsS(SliceRange,3) = smooth(OptCLineFunL(SliceRange),...
NewCLinePts(SliceRange,3),Param.SmtgCoef,'sgolay',Param.SmtgPolModel);
    % Do the "transition phase" to the non-smoothed points
    %-----------------------------------------------------
    NonSmoothedPts = NewCLinePts;
    Weight = linspace(0,1,Param.NPtsArndSmtg+2);
    Weight = Weight(2:end-1);
    % Points before the Neck
    c = 1;
    for i=Param.OptimSlices(1)-Param.NPtsArndSmtg:Param.OptimSlices(1)-1
        NonSmoothedPts(i,:) = ...
            (1-Weight(c))*NewCLinePts(i,:)+...
            Weight(c)*NewCLinePtsS(i,:);
        c = c+1;
    end
    % Points on the Neck
    NonSmoothedPts(Param.OptimSlices,:) =
NewCLinePtsS(Param.OptimSlices,:);
    % Points after the Neck
    c = 1;
    for
i=Param.OptimSlices(end)+1:Param.OptimSlices(end)+Param.NPtsArndSmtg
        NonSmoothedPts(i,:) = ...
            Weight(c)*NewCLinePts(i,:)+...
            (1-Weight(c))*NewCLinePtsS(i,:);
        c = c+1;
    end
    % Do a last smooting
    OptCLineFun1  = cscvn(NonSmoothedPts');
    OptCLineFun2L = LengthOfcscvn(OptCLineFun1,OptCLineFun1.breaks(1),...
                        OptCLineFun1.breaks,Param.LengthOfcscvnTol);
    OptCLineFun1_Pts = ppval(OptCLineFun1,OptCLineFun1.breaks);
    OptCLineFun2_PtsS(:,1) =
smooth(OptCLineFun2L,OptCLineFun1_Pts(1,:),...
```

```
        0.1,'sgolay',3);
    OptCLineFun2_PtsS(:,2) =
smooth(OptCLineFun2L,OptCLineFun1_Pts(2,:),...
        0.1,'sgolay',3);
    OptCLineFun2_PtsS(:,3) =
smooth(OptCLineFun2L,OptCLineFun1_Pts(3,:),...
        0.1,'sgolay',3);
    % Plot
    %--------------------------------------------------------------------
    Plot(NomCAS,OptCLineFun2_PtsS,1)
    % Prepare the Output
    %--------------------------------------------------------------------
    OptCLineFun  = cscvn(OptCLineFun2_PtsS');
end
end
function Plot(NomCAS,NewCLinePts,PlotFlag)
global Param
%% Plot the NewCoM, just for check:
%--------------------------------------------------------------------
if PlotFlag == 1
    figure('Position',Param.FigPos,'Name','SmoothMidLine',...
        'NumberTitle','off');
    plot3(NomCAS.Origin(:,1),NomCAS.Origin(:,2),NomCAS.Origin(:,3),'o-
b');
    hold on
    plot3(NewCLinePts(:,1),NewCLinePts(:,2),NewCLinePts(:,3),'*-r');
    legend({'Old CLine Pts','New CLine Pts'})
    grid on; axis equal; axis vis3d;
end

end
```

Stlread.m

```
function varargout = stlread(file)
disp('Stlread...')
% STLREAD imports geometry from an STL file into MATLAB.
%    FV = STLREAD(FILENAME) imports triangular faces from the binary STL
file
%    indicated by FILENAME, and returns the patch struct FV, with fields
'faces'
%    and 'vertices'.
%    [F,V] = STLREAD(FILENAME) returns the faces F and vertices V
separately.
%    [F,V,N] = STLREAD(FILENAME) also returns the face normal vectors.
%    The faces and vertices are arranged in the format used by the PATCH
plot
%    object.
%    Eric C. Johnson, 11-Dec-2008
%    Copyright 1999-2008 The MathWorks, Inc.
    if ~exist(file,'file')
        error(['File ''%s'' not found. If the file is not on MATLAB''s
path' ...
               ', be sure to specify the full path to the file.'], file);
    end
    fid = fopen(file,'r');
    if ~isempty(ferror(fid))
        error(lasterror); %#ok
    end
    M = fread(fid,inf,'uint8=>uint8');
    fclose(fid);
    if( isbinary(M) )
       [f,v,n] = stlbinary(M);
    else
       [f,v,n] = stlascii(M); % Not currently supported
    end
    varargout = cell(1,nargout);
    switch nargout
        case 2
            varargout{1} = f;
            varargout{2} = v;
        case 3
            varargout{1} = f;
            varargout{2} = v;
            varargout{3} = n;
        otherwise
            varargout{1} = struct('faces',f,'vertices',v);
    end
end
function [F,V,N] = stlbinary(M)
    F = [];
    V = [];
    N = [];
    if length(M) < 84
        error('MATLAB:stlread:incorrectFormat', ...
              'Incomplete header information in binary STL file.');
    end
    % Bytes 81-84 are an unsigned 32-bit integer specifying the number of
faces
```

```
    % that follow.
    numFaces = typecast(M(81:84),'uint32');
    %numFaces = double(numFaces);
    if numFaces == 0
        warning('MATLAB:stlread:nodata','No data in STL file.');
        return
    end
    T = M(85:end);
    F = NaN(numFaces,3);
    V = NaN(3*numFaces,3);
    N = NaN(numFaces,3);
    numRead = 0;
    while numRead < numFaces
        % Each facet is 50 bytes
        %  - Three single precision values specifying the face normal
vector
        %  - Three single precision values specifying the first vertex
(XYZ)
        %  - Three single precision values specifying the second vertex
(XYZ)
        %  - Three single precision values specifying the third vertex
(XYZ)
        %  - Two unused bytes
        i1    = 50 * numRead + 1;
        i2    = i1 + 50 - 1;
        facet = T(i1:i2)';
        n  = typecast(facet(1:12),'single');
        v1 = typecast(facet(13:24),'single');
        v2 = typecast(facet(25:36),'single');
        v3 = typecast(facet(37:48),'single');
        n = double(n);
        v = double([v1; v2; v3]);
        % Figure out where to fit these new vertices, and the face, in
the
        % larger F and V collections.
        fInd  = numRead + 1;
        vInd1 = 3 * (fInd - 1) + 1;
        vInd2 = vInd1 + 3 - 1;

        V(vInd1:vInd2,:) = v;
        F(fInd,:)        = vInd1:vInd2;
        N(fInd,:)        = n;
        numRead = numRead + 1;
    end
end
function [F,V,N] = stlascii(A) %#ok
    warning('MATLAB:stlread:ascii','ASCII STL files currently not
supported.');
    F = [];
    V = [];
    N = [];
end
function tf = isbinary(A)
% ISBINARY determines if an STL file is binary or ASCII.
    % Look for the string 'endsolid' near the end of the file
    if isempty(A) || length(A) < 16
        error('MATLAB:stlread:incorrectFormat', ...
              'File does not appear to be an ASCII or binary STL file.');
```

```
    end
    % Read final 16 characters of M
    i2  = length(A);
    i1  = i2 - 15;
    str = char( A(i1:i2)' );

    k = strfind(lower(str), 'endsolid');
    if ~isempty(k)
        tf = false; % ASCII
    else
        tf = true;  % Binary
    end
end
```

TimeOfcscvn.m

```
function t2 = TimeOfcscvn(Fun,t0,tmax,Len,tol)
for i =1:length(Len)
    ToOpt =@(t1) abs(Len(i)-LengthOfcscvn(Fun,t0,t1,tol));
%     t2(i) = fminsearch(ToOpt,t0+100);
    t2(i) = fminbnd(ToOpt,t0,tmax);
end
```

ANNEX B. Produce1DStraightStent.m

```
%% Create a straight stent:
%-------------------------------------------------------------------------
RStent = 2;
LStent = 10;
RadiusOffset = 0;
NTurns =1.5;
NptsOnMidLine = 100; %10-100;
Coil_NrOf = 10;
RadiusFigFlag = 1;
StentFigFlag = 1;
WriteMidPts = 0;
WriteCoilPts = 0;
%% Produce the straight middle line
%-------------------------------------------------------------------------
AllMidLinePts = [zeros(NptsOnMidLine,1),...
zeros(NptsOnMidLine,1),...
linspace(-0.5*LStent, 0.5*LStent, NptsOnMidLine)'];
%% Initialisations
%-------------------------------------------------------------------------
% Figures
scrsz = get(0,'ScreenSize')+[0,50,0,0];
FigWidth = (scrsz(3)-scrsz(1))/3;
FigHeight = (scrsz(4)-scrsz(2))/2;
if RadiusFigFlag == 1
Radius_H = figure('OuterPosition',...
[0*FigWidth,scrsz(2) + 1*FigHeight,...
FigWidth, FigHeight]);
end
if StentFigFlag == 1
Stent_H = figure('OuterPosition',...
[1*FigWidth,scrsz(2) + 1*FigHeight,...
FigWidth, FigHeight]);
end
Curve = cell(1, Coil_NrOf);
% Angles and Turning Direction
%-------------------------------------------------------------------------
if Coil_NrOf > 1
LRHelix = [0*ones(1,Coil_NrOf/2),1*ones(1,(Coil_NrOf/2))];
Alpha   = 2*pi*[linspace(0,(Coil_NrOf-2)/Coil_NrOf,Coil_NrOf/2),...
linspace(0,(Coil_NrOf-2)/Coil_NrOf,Coil_NrOf/2)+...
0.5*(Coil_NrOf-2)/Coil_NrOf];
else
LRHelix = 1;
Alpha   = 0;
end
for i=1:NptsOnMidLine
tnorm = i/NptsOnMidLine;
% Do the points on the first plane for all coils
%-------------------------------------------------------------------------
for j=1:size(Alpha,2)
% Produce the poins on the z=AllMidLinePts(i,3)-plane
if LRHelix(j)==1
PtToBeProj = [...
(RStent+RadiusOffset)*cos(Alpha(j)+NTurns*tnorm*2*pi),...
```

```
(RStent+RadiusOffset)*sin(Alpha(j)+NTurns*tnorm*2*pi),...
AllMidLinePts(i,3)];
else
PtToBeProj = [...
(RStent+RadiusOffset)*cos(-Alpha(j)-NTurns*tnorm*2*pi),...
(RStent+RadiusOffset)*sin(-Alpha(j)-NTurns*tnorm*2*pi),...
AllMidLinePts(i,3)];
end
% Register the points of the struts
%-------------------------------------------------------------------
Curve{j}(:,i) = PtToBeProj;
end
end
% Plot the struts
%-------------------------------------------------------------------------
figure(Stent_H);
for c_1 = 1:size(Curve,2)
plot3(Curve{c_1}(1,:),Curve{c_1}(2,:),Curve{c_1}(3,:),'-k')
hold on
end
% Plot The Midline
%-------------------------------------------------------------------------
plot3(AllMidLinePts(:,1),AllMidLinePts(:,2),AllMidLinePts(:,3),'.r')
% Write CoilPts File
if WriteCoilPts == 1
WriteCoilPtsFile(CoilPtsName, Curve)
end
if WriteMidPts == 1
WriteMidPtsFile(MidlinePtsName, AllMidLinePts)
end
% Plot Cosmetics
%-------------------------------------------------------------------------
xlabel('x')
xlabel('y')
xlabel('z')
grid on
axis equal
```

ANNEX C. Code to test the results

```
% 1.comapre the two centerlines:
% ptscenterline1=greenstent(real)-red (with aneurysm)
% ptscenterline2=pinkstent(ideal)-blue (without aneurysm)
ptscenterline1 = ppval(MidLineFun, MidLineFun.breaks(14:end-6)); %poso
entre 14 i end-6 pq centrelines de diferent longitud
MidLineFun1    = cscvn(ptscenterline1);
ptscenterline2 = ppval(MidLineFun2,MidLineFun2.breaks);
dist       = [];
dist_error = [];
Iimp       = [];
% plot the two centerlines:
figure(1)
plot3(ptscenterline1(1,:),ptscenterline1(2,:),ptscenterline1(3,:),'.-r');
hold on;
% plot3(ptscenterline2(1,:),ptscenterline2(2,:),ptscenterline2(3,:),'*-
b'); hold on;
title('Plot the two centerlines')
axis equal
%% Find the length of the centerline blue:
InterpolationPts_NrOf = 50;
InterpolationTimes    =
linspace(MidLineFun2.breaks(1),MidLineFun2.breaks(end),InterpolationPts_N
rOf);
% InterpolationPts      = ppval(MidLineFun1,InterpolationTimes);
InterpolationPts2     = ppval(MidLineFun2,InterpolationTimes);
%% Visualisation (Plot your interpolationPts):
hold on
plot3(InterpolationPts2(1,:),...
      InterpolationPts2(2,:),...
      InterpolationPts2(3,:),'.-k')
% Create a plane that contains one point of the 'ideal' centerline:
[T,N,B,k,t] =
frenet(InterpolationPts2(1,:),InterpolationPts2(2,:),InterpolationPts2(3,
:));
 for i = 1:size(InterpolationPts2,2)
     for j = 1:size(ptscenterline1,2) %-1
         if j <= 31
             [I,check] =
plane_line_intersect(T(i,:),InterpolationPts2(:,i),ptscenterline1(:,j),pt
scenterline1(:,j+1));
             if check == 1 || check==2
                 Iimp = [Iimp;I']; % points that are in the plane and in
the red line
             end
         end
     end
 end
% Distance point-point:
 dist1 = Inf; % dist1 is a tolerance
 counter = [];
 for i = 1:size(InterpolationPts2,2)
     for j = 1:size(Iimp)
         dist = sqrt((InterpolationPts2(1,i)-
Iimp(j,1))^2+(InterpolationPts2(2,i)-
```


```
Iimp(j,2))^2+(InterpolationPts2(3,i)-Iimp(j,3))^2);
         if dist <= dist1
             dist1 = dist;
             counter(i) = j;
         end
     end
     dist_error = [dist_error; dist1];
     dist1      = Inf;
 end
[arclen,seglen] =
arclength(InterpolationPts2(1,:),InterpolationPts2(2,:),InterpolationPts2
(3,:));
Laxis = zeros(50,1);
Laxis(2:end,1) = seglen;
val = 0;
for k = 1:size(Laxis)
    Xaxis(k) = Laxis(k) + val;
    val = Xaxis(k);
end
figure(2)
plot(Xaxis,dist_error,'*-b','LineWidth',3); hold on; grid on;
set(gca,'LineWidth',3)
set(gca,'Fontsize',20)
set(get(gca,'XLabel'),'FontSize',15)
title('Plot the distance between the two centerlines')
xlabel('Length along the Centerline [mm]')
ylabel('Distance between the two centerlines [mm]')
axis normal;
%% 2.compare the RadiusFunction:
DistRadError = [];
Rad1 = ppval(RadFun, InterpolationTimes);
Rad2 = ppval(RadFun2, InterpolationTimes);
for i = 1:InterpolationPts_NrOf
    DistRadError = [DistRadError; abs(Rad2(1,i)-Rad1(1,i))];
end
figure(3)
plot(Xaxis,DistRadError,'*-r','LineWidth',3); hold on; grid on;
set(gca,'LineWidth',3)
set(gca,'Fontsize',20)
set(get(gca,'XLabel'),'FontSize',15)
title('Compare the minimum Radius')
xlabel('Length along the Centerline [mm]')
ylabel('Difference between Radius  [mm]')
axis normal;
%% See all to compare:
% figure(4)
% subplot(4,1,1)
% plot(Xaxis,dist_error,'-g','LineWidth',3); hold on; grid on;
% set(gca,'LineWidth',3)
% set(gca,'Fontsize',10)
% set(get(gca,'XLabel'),'FontSize',15)
% % title('Plot the distance between the two centerlines')
% xlabel('Length along the Centerline [mm]')
% ylabel('Distance between the two centerlines [mm]')
% subplot(4,1,2)
% plot(Xaxis,DistRadError,'*-r','LineWidth',3); hold
on;plot(Xaxis,DistRadErrorMax,'*-r','LineWidth',3); hold on; grid on;
% set(gca,'LineWidth',3)
```

```
% set(gca,'Fontsize',10)
% set(get(gca,'XLabel'),'FontSize',15)
% % title('Plot the RadFun1')
% xlabel('Length along the Centerline [mm]')
% ylabel('MinRad-R & MaxRad-R [mm]]')
% subplot(4,1,3)
% plot(Xaxis,Rad2,'-b','LineWidth',3); hold on; grid on;
% set(gca,'LineWidth',3)
% set(gca,'Fontsize',10)
% set(get(gca,'XLabel'),'FontSize',15)
% % title('Plot the RadFun2')
% xlabel('Length along the Centerline [mm]')
% ylabel('Radius Real Stent [mm]')
% subplot(4,1,4)
% plot(Xaxis,DistRadError,'-k','LineWidth',3); hold on; grid on;
% set(gca,'LineWidth',3)
% set(gca,'Fontsize',10)
% set(get(gca,'XLabel'),'FontSize',15)
% % title('Plot the Radius Error')
% xlabel('Length along the Centerline [mm]')
% ylabel('Difference between Radius [mm]')
%% 3.Compare the ovality:
MinRad2 = ppval(RadFun2, InterpolationTimes);
MaxRad2 = ppval(RadFunMax, InterpolationTimes);
for i = 1:size(MinRad2,2)
    Ovality1(i) = MinRad2(i)/Rad1(i);
    Ovality2(i) = MaxRad2(i)/Rad1(i);
    Ovality3(i) = MinRad2(i)/MaxRad2(i);
end
figure(7)
plot(Xaxis, Ovality1,'*-r','LineWidth',3); hold on; grid on;
set(gca,'LineWidth',3)
set(gca,'Fontsize',20)
set(get(gca,'XLabel'),'FontSize',15)
title('Plot the Radius ratio = MinRad/R')
xlabel('Length along the Centerline [mm]')
ylabel('Radius ratio [-]')
figure(8)
plot(Xaxis, Ovality2,'*-b','LineWidth',3); hold on; grid on;
set(gca,'LineWidth',3)
set(gca,'Fontsize',20)
set(get(gca,'XLabel'),'FontSize',15)
title('Plot the Radius ratio = MaxRad/R')
xlabel('Length along the Centerline [mm]')
ylabel('Radius ratio [-]')
% figure(9)
% plot(Xaxis, Ovality3,'*-k','LineWidth',3); hold on; grid on;
% set(gca,'LineWidth',3)
% set(gca,'Fontsize',20)
% set(get(gca,'XLabel'),'FontSize',15)
% title('Plot the Radius ratio3 = MinRad/MaxRad')
% xlabel('Length along the Centerline [mm]')
% ylabel('Radius ratio3 [-]')
%% Plot The MinRad & MaxRad & Rad1 just to check that we are comparing
the same slides
figure(10)
% subplot(3,1,1)
% plot(Xaxis, MinRad2,'-b','LineWidth',3); hold on; grid on;
```

```
% set(gca,'LineWidth',3)
% set(gca,'Fontsize',20)
% set(get(gca,'XLabel'),'FontSize',15)
% % title('Plot the Minimum Radius')
% xlabel('Length along the Centerline [mm]')
% ylabel('MinRad [mm]')
% subplot(3,1,2)
% plot(Xaxis, MaxRad2,'-r','LineWidth',3); hold on; grid on;
% set(gca,'LineWidth',3)
% set(gca,'Fontsize',20)
% set(get(gca,'XLabel'),'FontSize',15)
% % title('Plot the Maximum Radius')
% xlabel('Length along the Centerline [mm]')
% ylabel('MaxRad [mm]')
% subplot(3,1,3)
plot(Xaxis, MinRad2,'*','LineWidth',3); hold on; plot(Xaxis,
MaxRad2,'*r','LineWidth',3); hold on; plot(Xaxis,
Rad1,'*g','LineWidth',3); hold on;
set(gca,'LineWidth',3)
set(gca,'Fontsize',20)
set(get(gca,'XLabel'),'FontSize',15)
title('Plot the all Radius Points: MinRad, MaxRad and R')
xlabel('Length along the Centerline [mm]')
ylabel('All Radius [mm]')
% Plot the two ovalities:
% figure(11)
% subplot(2,1,1)
% plot(Xaxis, Ovality1,'*-r','LineWidth',3); hold on; grid on;
% set(gca,'LineWidth',3)
% set(gca,'Fontsize',20)
% set(get(gca,'XLabel'),'FontSize',15)
% title('Plot the Radius ratio = MinRad/R')
% xlabel('Length along the Centerline [mm]')
% ylabel('Radius ratio [-]')
% subplot(2,1,2)
% plot(Xaxis, Ovality2,'*-b','LineWidth',3); hold on; grid on;
% set(gca,'LineWidth',3)
% set(gca,'Fontsize',20)
% set(get(gca,'XLabel'),'FontSize',15)
% title('Plot the Radius ratio2 = MaxRad/R')
% xlabel('Length along the Centerline [mm]')
% ylabel('Radius ratio [-]')
DistRadErrorMax = [];
RadMax = ppval(RadFunMax, InterpolationTimes);
for i = 1:InterpolationPts_NrOf
    DistRadErrorMax = [DistRadErrorMax; abs(RadMax(1,i)-Rad1(1,i))];
end
figure(11)
plot(Xaxis,DistRadError,'*-r','LineWidth',3); hold
on;plot(Xaxis,DistRadErrorMax,'*-b','LineWidth',3); hold on; grid on;
set(gca,'LineWidth',3)
set(gca,'Fontsize',20)
set(get(gca,'XLabel'),'FontSize',15)
title('Plot MinRad-R & Plot MaxRad-R')
xlabel('Length along the Centerline [mm]')
ylabel('MinRad-R & MaxRad-R [mm]')
figure(4)
subplot(3,1,1)
```

```
plot(Xaxis,dist_error,'-g','LineWidth',3); hold on; grid on;
set(gca,'LineWidth',3)
set(gca,'Fontsize',15)
set(get(gca,'XLabel'),'FontSize',15)
title('Plot the distance between the two centerlines')
% xlabel('Length along the Centerline [mm]')
ylabel('Distance btwn the 2 centerlines [mm]')
subplot(3,1,2)
plot(Xaxis,DistRadError,'*-r','LineWidth',3); hold
on;plot(Xaxis,DistRadErrorMax,'*-b','LineWidth',3); hold on; grid on;
set(gca,'LineWidth',3)
set(gca,'Fontsize',15)
set(get(gca,'XLabel'),'FontSize',15)
title('Plot the MinRad-R & MaxRad-R')
% xlabel('Length along the Centerline [mm]')
ylabel('MinRad-R & MaxRad-R [mm]]')
subplot(3,1,3)
plot(Xaxis, Ovality1,'*-r','LineWidth',3); hold on; plot(Xaxis,
Ovality2,'*-b','LineWidth',3); hold on; grid on;
set(gca,'LineWidth',3)
set(gca,'Fontsize',15)
set(get(gca,'XLabel'),'FontSize',15)
title('Plot the Radius ratios')
xlabel('Length along the Centerline [mm]')
ylabel('Radius ratios [-]')
```

ANNEX D. Distance Between Point and Triangle in 3D

```
function [dist point]=distPointTriangle(p,v1,v2,v3)
% according to
% www.geometrictools.com/Documentation/DistancePoint3Triangle3.pdf
%Calculem l'area com a mesura preventiva
area=0.5*norm(cross(v2-v1,v3-v1));
% Descartem el triangle si te l'area massa petita (triangle degenerat)
if(area<0.01)
dist=min([norm(p-v1),norm(p-v2),norm(p-v3)]');
point=[0;0;0];
return;
end
P=p;
B=v1; %origen of the triangle plane
E0=v2-v1;
E1=v3-v1;
D=B-P;
a=dot(E0,E0);
b=dot(E0,E1);
c=dot(E1,E1);
d=dot(E0,D);
e=dot(E1,D);
f=dot(D,D);
det=a*c-b*b; s=b*e-c*d; t=b*d-a*e;
% Region classification
if (s+t <= det)
if (s<0)
if (t<0)
region=4;
else
region=3;
end
else
if (t<0)
region=5;
else
region=0;
end
end
else
if(s<0)
region=2;
else
if(t<0)
region=6;
else
region=1;
end
end
end
% compute distance according the region
switch(region)
case 0
```

```
s=s/det;
t=t/det;
case 1
numer=c+e-b-d;
if (numer <=0)
s=0;
else
denom=a-2*b+c;
if(numer>=denom)
s=1;
else
s=numer/denom;
end
end
t=1-s;
case 3
s=0;
if (e>=0)
t=0;
else
if(-e>=c)
t=1;
else
t=-e/c;
end
end
case 5
t=0;
if (d>=0)
s=0;
else
if(-d>=a)
s=1;
else
s=-d/a;
end
end
case 2
tmp0=b+d;
tmp1=c+e;
if (tmp1>tmp0)
numer=tmp1-tmp0;
denom=a-2*b+c;
if (numer >=denom)
s=1;
else
s=numer/denom;
end
t=1-s;
else
s=0;
if(tmp1<=0)
t=1;
else
if(e>=0)
t=0;
else
t=-e/c;
```

```
end
end
end
case 6
tmp0=a+d;
tmp1=b+e;
if (tmp1>tmp0)
numer=tmp1-tmp0;
denom=a-2*b+c;
if (numer >=denom)
s=1;
else
s=numer/denom;
end
t=1-s;
else
t=0;
if(tmp1<=0)
s=1;
else
if(e>=0)
s=0;
else
s=-e/b;
end
end
end
case 4
if(d<=0)
t=0;
if(a<-d)
s=0;
else
s=-d/a;
end
else
s=0;
if(c<-e)
t=0;
else
t=-c/e;
end
end
end
dist=norm(P-B-s*E0-t*E1);
point=B+s*E0+t*E1;
%[s t dist]
```