vmtktools_docs Documentation

Release 0.1

Henrik Kjeldsberg

Sep 13, 2018

CONTENTS

This is experimental documentation for the VMTK Tools. This version of the documentation on Read the Docs is under development.

CHAPTER

ONE

INSTALLATION

This guide summarises how to install VMTK Tools. Currently the project is only accessable through github.

1.1 Compatibility and Dependencies

The general dependencies of VMTK Tools are

- ITK 4.13
- VTK 8.1
- H5Py
- Numpy <= 1.13
- VMTK 1.4

VMTK no longer support python 2.7 on Windows builds. VMTK 1.4+ requires python 3.5+ on Windows 10. Linux and MacOSX machines support python 2.7, 3.5+.

1.2 Basic Installation

To install the Python components of VMTK Tools:

```
git clone https://github.com/aslakbergersen/vmtktools
git clone https://github.com/hkjeldsberg/automatedManipulation
```

Todo: Update VMTK Tools documentation for lates release, include test examples, and provide a link to documentation of stable version here.

CHAPTER

TWO

USING VMTK TOOLS

2.1 Application

VMTK Tools is a collection of scripts to objectivly manipulate geometric features of patient-specific vascular geometries. In this tutorial we examplify the usage by manipulating internal carotid arteries (wiki link), but the tool can be applied to any geometry with a tubular shape. The geometries used here were

Centence from removal: on geometries

The algorithms presented in the framework was originally proposed and implemented in Bergersen's and Kjeldsberg's master theses (2016, 2018). TODO: Add link.

[//]: <> (The goal with VMTK Tools is to provide researchers with a set of tools to manipulate one specific geometric feature at the time to to better understand the impact of geometry in) [//]: <> (When the JOSS paper is published, add a note that it should be cited) [//]: <> (When the method paper is published, add a note that it should be cited)

[//]: <> (This tutorial illustrates the steps for the automated geometric manipulation in patient-spesific morphologies of carotid arteries.)

Manipulating surfaces is no trivial task, therefore we derive an alternative representation All the algorithms are based on lgorithm all utilizes Voronoi Diagrams and on its properties, particularly on the fact that given the model surface its Voronoi Diagram can be derived and vice versa.

Give credit where credit it due: Thank the VTK, VMTK projects, and Ford et al. from where we gathered these ideas.

2.2 Definitions

Inlet seed

2.3 Initialize cases

Prior to manipulation, the models need to be initialized by computing their centerline and Voronoi diagram. The goal of the initialization is to achieve three separate centerlines:

1. Two single centerlines through each daughter vessel
2. A single centerline between the daughter vessels
3. The complete network of centerlines through the model

To initialize, run the `initialize_cases.py` file:

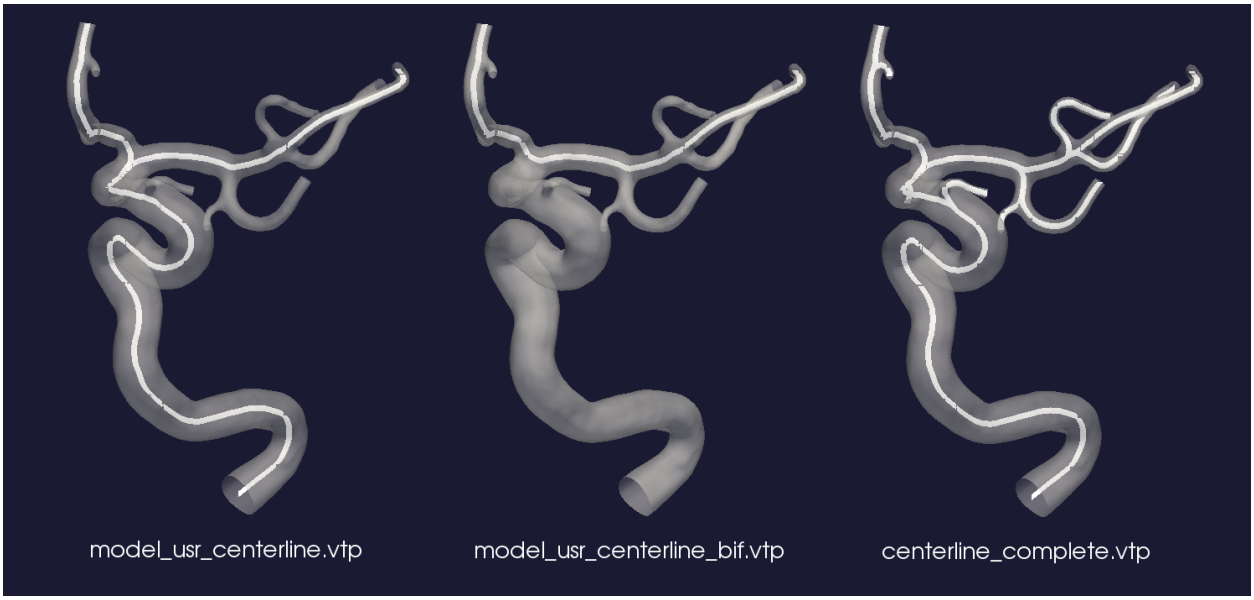

Fig. 1: Figure 1: Expected outcome of initialization, using a representative model

```
python initialize_cases.py --dir_path PATH_TO_CASES
```

Running this command will cause a render window to appear, asking you to specify points on the surface which will act as source points. This will trigger the following prompt:

```
Please position the mouse and press space to add source points, 'u' to undo
```

To select the source, click on the mesh and a red sphere should appear. When the source has been selected, press q to continue. Now you'll be prompted:

```
Please position the mouse and press space to add target points, 'u' to undo
```

After selecting target points, press q to continue and the centerlines between the selected points will be computed (may take some time). By performing these steps three times, you should have the centerlines specified above. This concludes the initialization.

2.4 Landmarking

Landmarking of the geometry is performed in order to identify different segments of the vessel. Identification of specific segments is required in order to perform geometric manipulation. The script `automated_landmarking.py` includes implementation of two automated landmarking algorithms; the first introduced by Piccinelli et al. (2011), the second by Bogunović et al. (2014).

Both landmarking algorithms rely on geometric properties of the centerline. The script allows the user to select one of four different methods to compute the curvature of the discrete centerline:

1. B-Splines (`spine`)
2. Free-knot regression splines (`freeknot`)
3. Discrete derivatives (`disc`)
4. VMTK (`vmtk`)

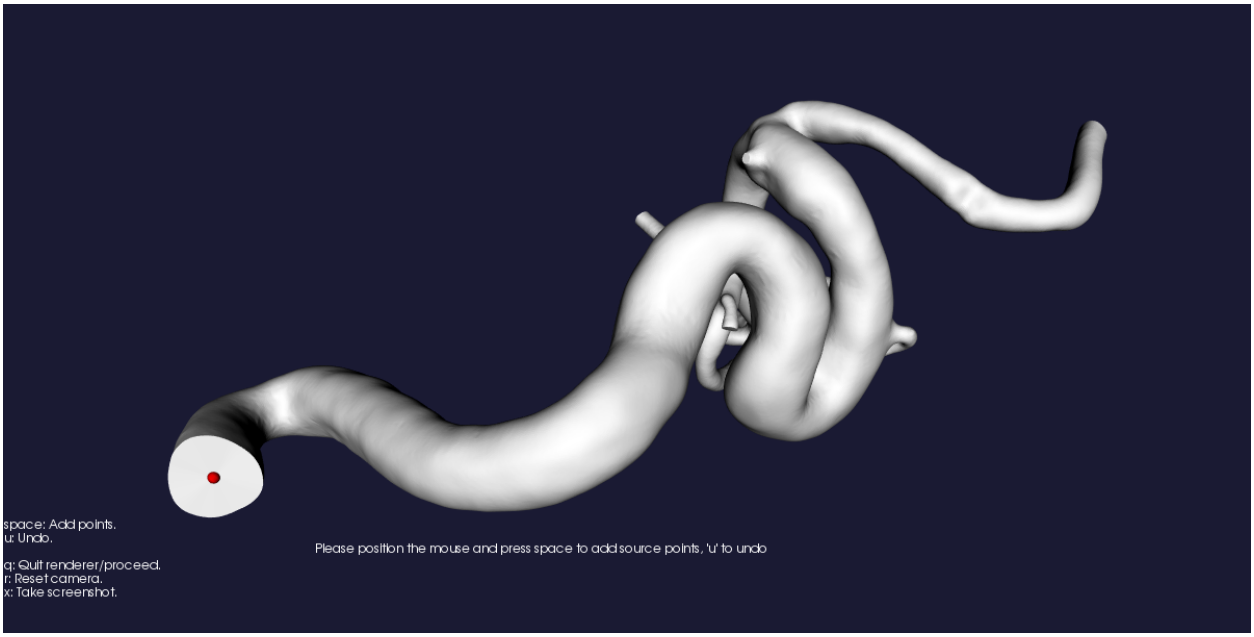

Fig. 2: Figure 2: Placing source seeds.

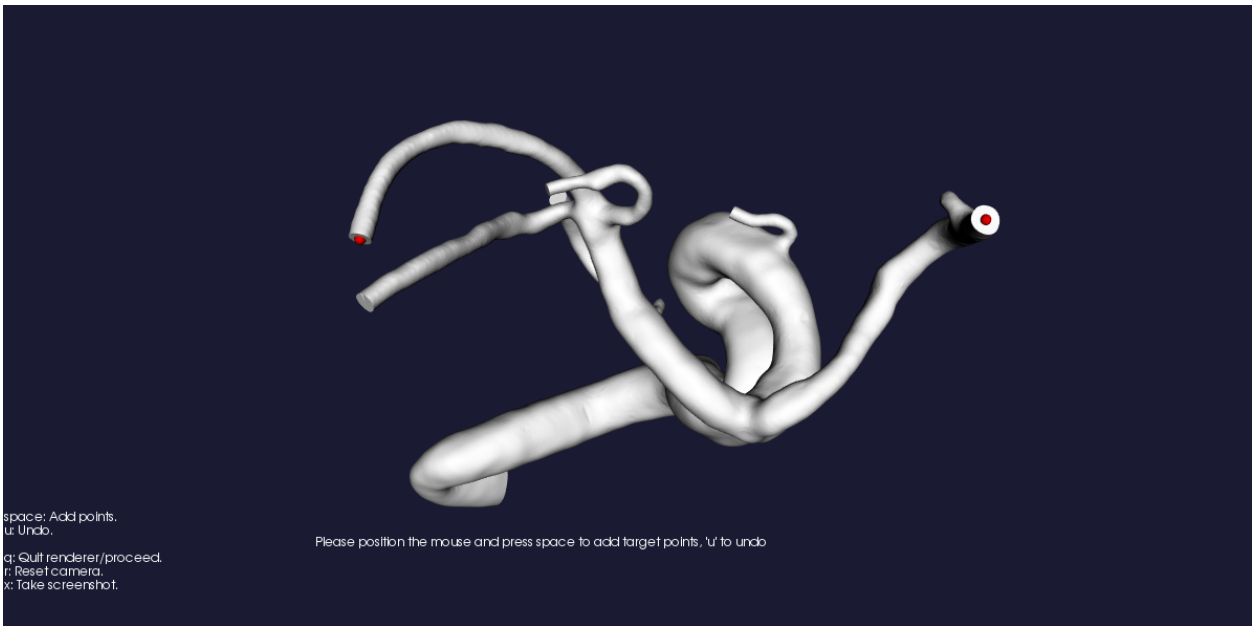

Fig. 3: Figure 3: Placing target seeds.

Todo: Add references to different splining techniques?

To perform landmarking, run the following command:

```
python automated_landmarking.py --dir_path [PATH_TO_CASES] --algorithm [ALGORITHM] --
↪curv_method [METHOD]
```

This will produce `landmark_ALGORITHM_CURVMETHOD.particles` which contains four points defining the interfaces between the segments of the vessel.

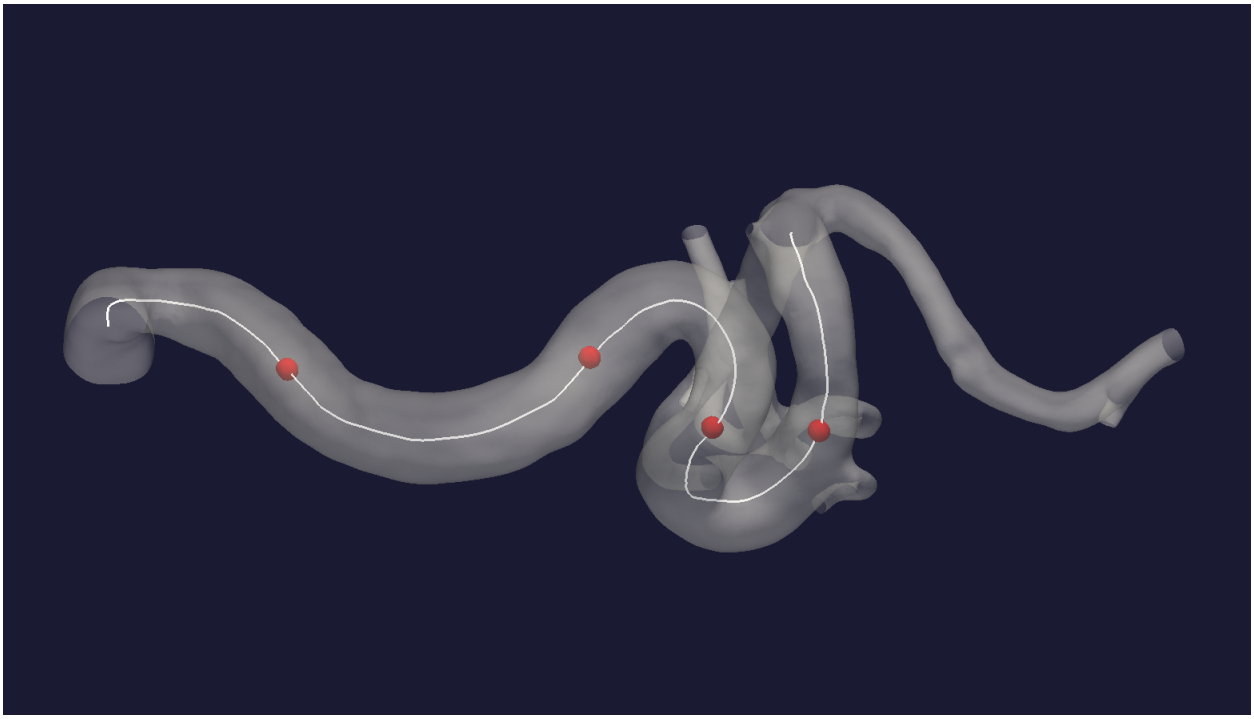

Fig. 4: Figure 4: Landmarked geometry, with interfaces shown as red spheres.

2.5 Geometric manipulation

The framework presented here allows for geometric manipulation of four independant morphological features of the carotid arteries.

- Bifurcation angle rotation
- Cross-sectional area variation
- Curvature variation in the anterior bend
- Angle variation in the anterior bend

2.5.1 Bifurcation angle rotation

The script `move_branches.py` performes an objective rotation of two daughter branches, moving them a given angle along the bifurcation plane. In this implementation, positive and negative angles rotate the branches upward

and downward, respectively. The implementation rotates both branches by default, but the user can specificy that only rotate a single branch. This is achieved by setting the argument `keep-fixed1` or `keep-fixed2` to **True**, to leave the first or second branch, respectively.

To perform rotation of the daughter branches, run the following generalized command:

```
python move_branches.py --dir_path [PATH_TO_CASES] --case [CASENAME] --angle [ANGLE] -
↪-lower True
```

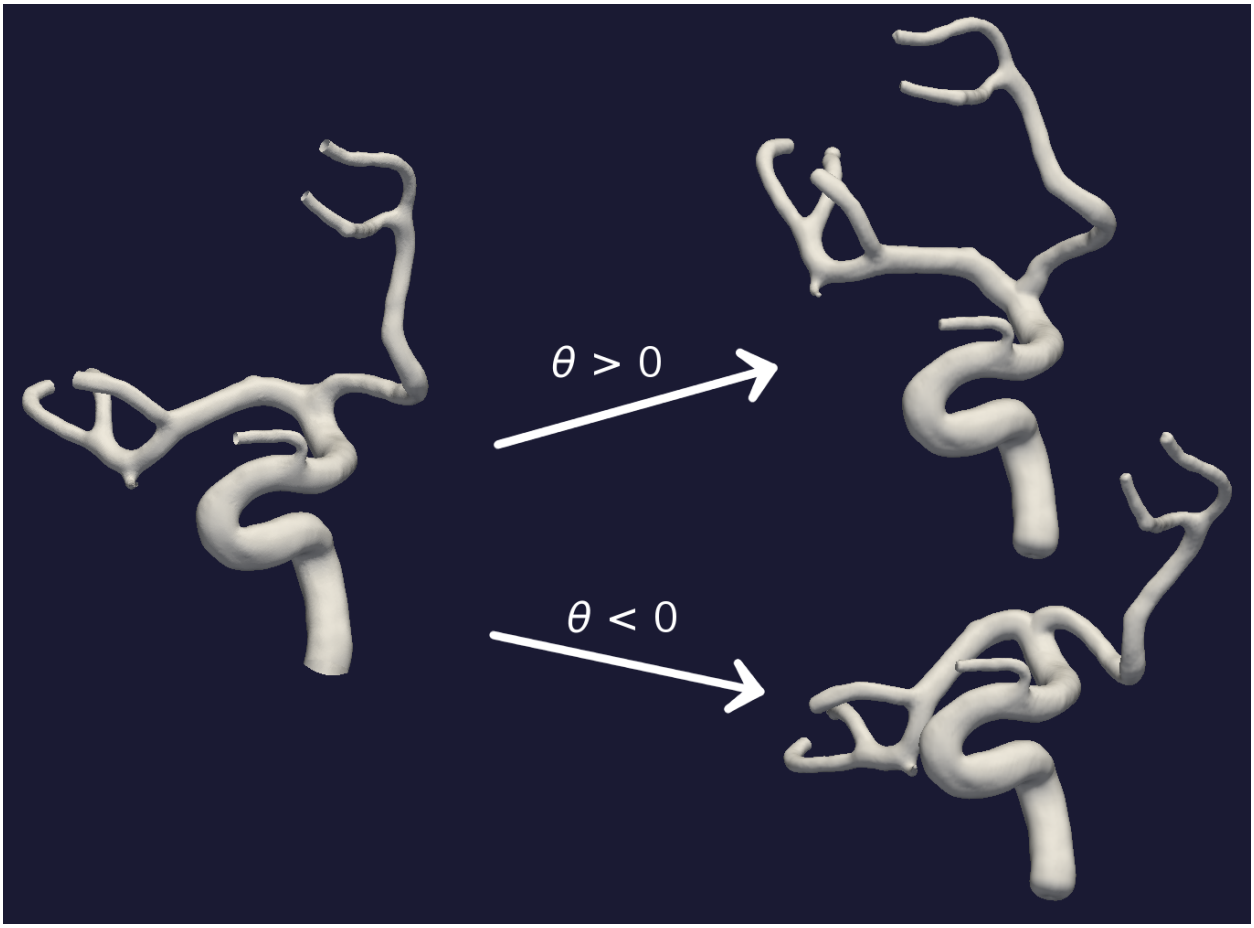

Fig. 5: Figure 5: Rotation of daughter branches, in both a widening and narrowing of the bifurcation angle.

In addition, the user includes two options for reconstruction of the bifurcation, determined by the parameters `bif` and `lower` set to **True**. The `lower` parameter creates a more realistic looking bifurcation, while the `bif` parameter creates a straight segment between the daughter branches. [\]: <> (Cite or refer to reconstruction paper) A comparison is shown below, where the straight segment is created by running:

```
python move_branches.py --dir_path [PATH_TO_CASES] --case [CASENAME] --angle [ANGLE] -
↪-bif True
```

2.5.2 Cross-sectional area variation

Manipulation of the cross-sectional area is performed by running the script `area_variations.py`. [\]: <> (The script performs geometric manipulation of the cross sectional area, represented by a centerline.) In order to preserve the inlet and the end of the geometry segment, the first and last 10% of the area of interest are left unchange.

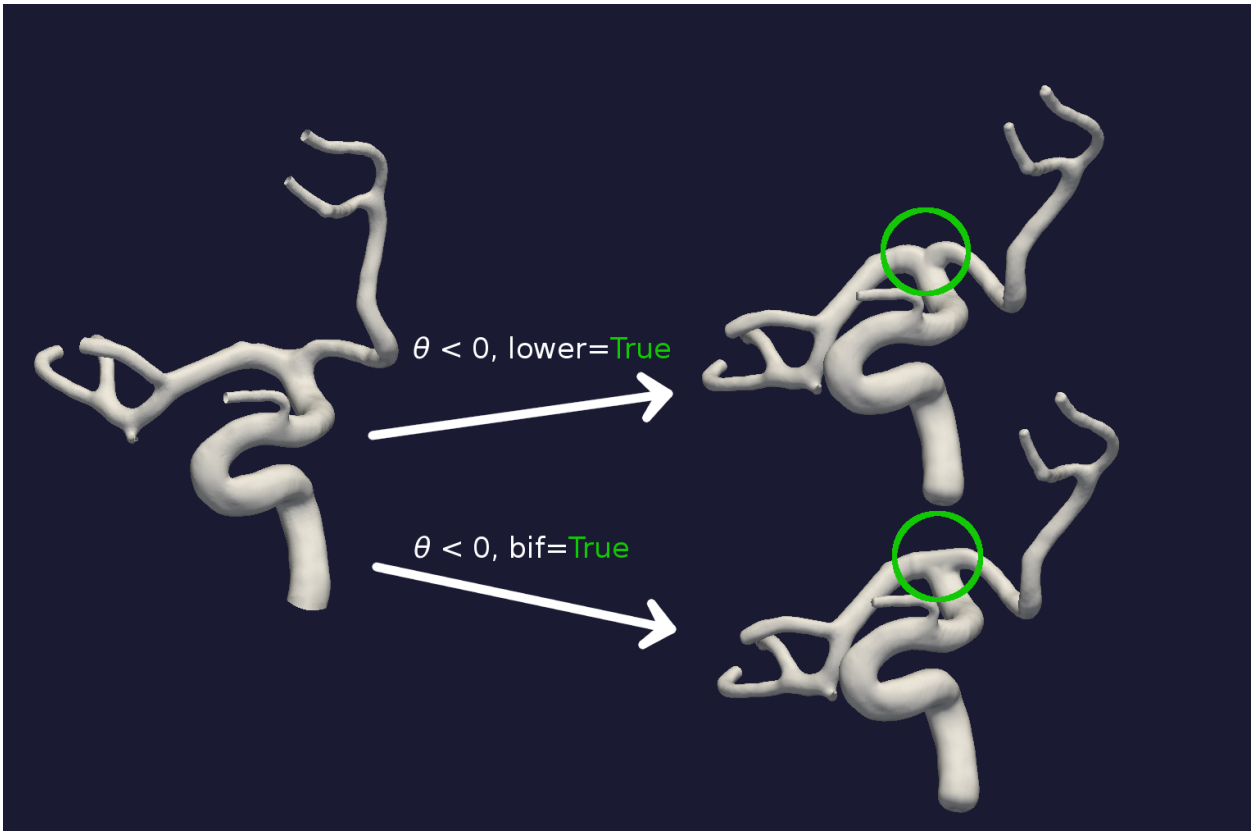

Fig. 6: Figure 6: Rotation of daughter branches with different reconstruction of the bifurcation.

The script `area_variations.py` includes several options for area variations:

- Area variations
- In-/deflation
- Stenosis creation/removal

Area variations

Tulle illustrasjon av geometrien

Focus on R, and not beta.

Area variation is initialized by a factor β or a ratio, $R = A_{min}/A_{max}$. The script takes `beta` and `ratio` as command line arguments. However, the script requires only one of the two parameters to perform area variation. Setting $\beta < 0$ will cause the ratio R to decrease, whereas $\beta > 0$ will cause the ratio to increase. The ratio, R, behaves as shown in the illustration below.

Examplified in Figure 3, where the ICA is defined as the region of interest.

To perform area variations of the vessel area, run the following command:

```
python area_variations.py --dir_path [PATH_TO_CASES] --case [CASENAME] --smooth True -
↪-beta 0.5
```

or:

```
python area_variations.py --dir_path [PATH_TO_CASES] --case [CASENAME] --smooth True -
↪-ratio 0.5
```

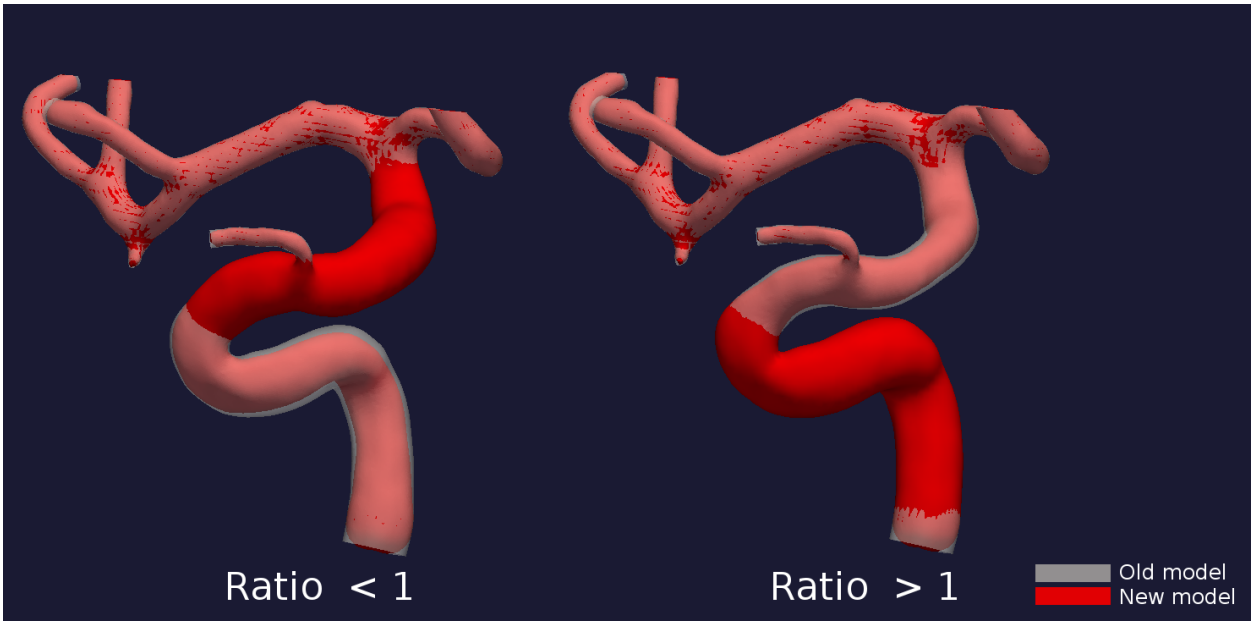

Fig. 7: Figure : Area variations throughout the geometry for different ratios.

Comment to the Figure: Old, new –> original and manipulated.

Overall area variation

The area of interest can also be increase or decreased overall, by using the `percentage` argument. The `percentage` argument determines the percentage to increase or decrease the overall area.

To perform overall increase or decrease of the area of interest, run the following command:

```
python area_variations.py --dir_path [PATH_TO_CASES] --case [CASENAME] --smooth True -
↪-percentage [%]
```

Below is an illustration of area decrease and increase in a single patient-specific model.

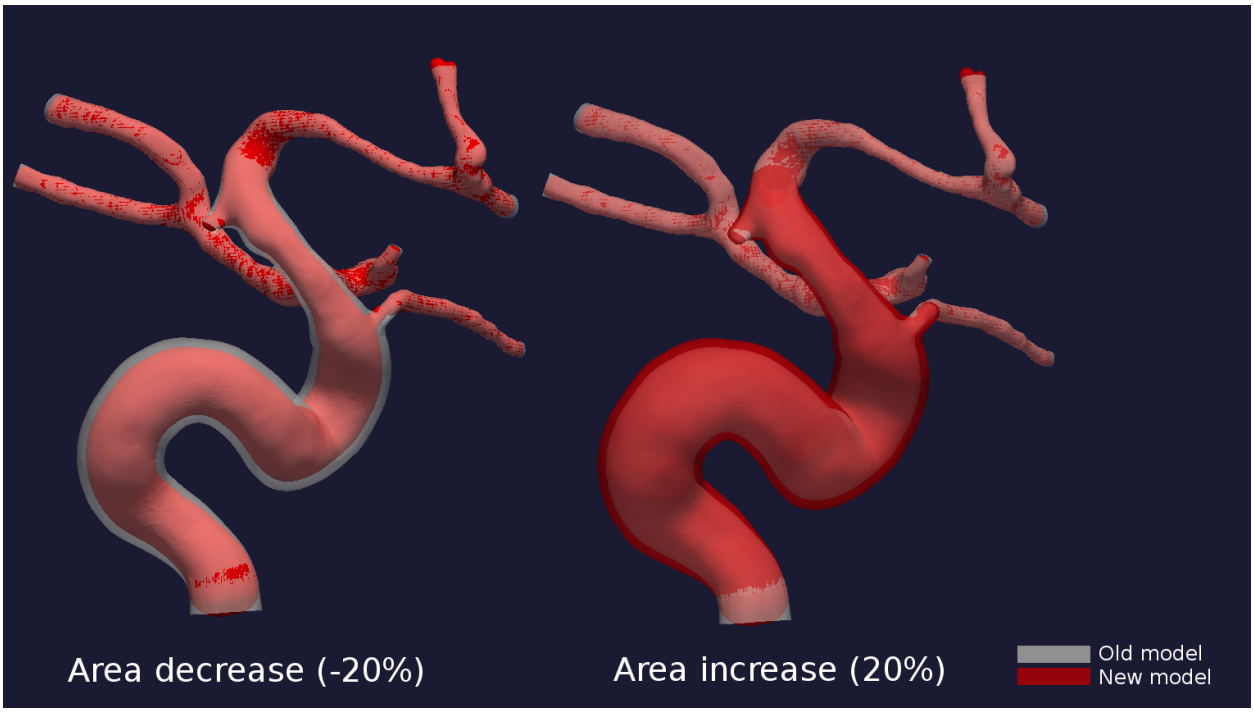

Fig. 8: Figure : Decrease and increase in overall area.

Stenosis creation / removal

The framework allows for creation or removal of one stenosis located along the area of interest. Creation and removal of a stenosis is performed by specifying input argument `stenosis` to **True**. Spesifically for stenosis creation, the input arguments `percentage` and `length` determine the area reduction and length of the stenosis, respectively. The `length` argument is multiplied with the radius at the selected center point of the stenosis, to expand the stenosis exposed area.

Comment KVS: Upstream and downstream. Write sine function, can easly be modified in the script.

For creation of a stenosis, run the follwing command:

```
python area_variations.py --dir_path [PATH_TO_CASES] --case [CASENAME] --smooth True -
↪-percentage [%] --stenosis True --size [SIZE]
```

Running this command will cause a render window to appear, asking (replace!) you to specify points on the surface which will act as the center point of the stenosis. This will trigger the following prompt, with a suggested point placement:

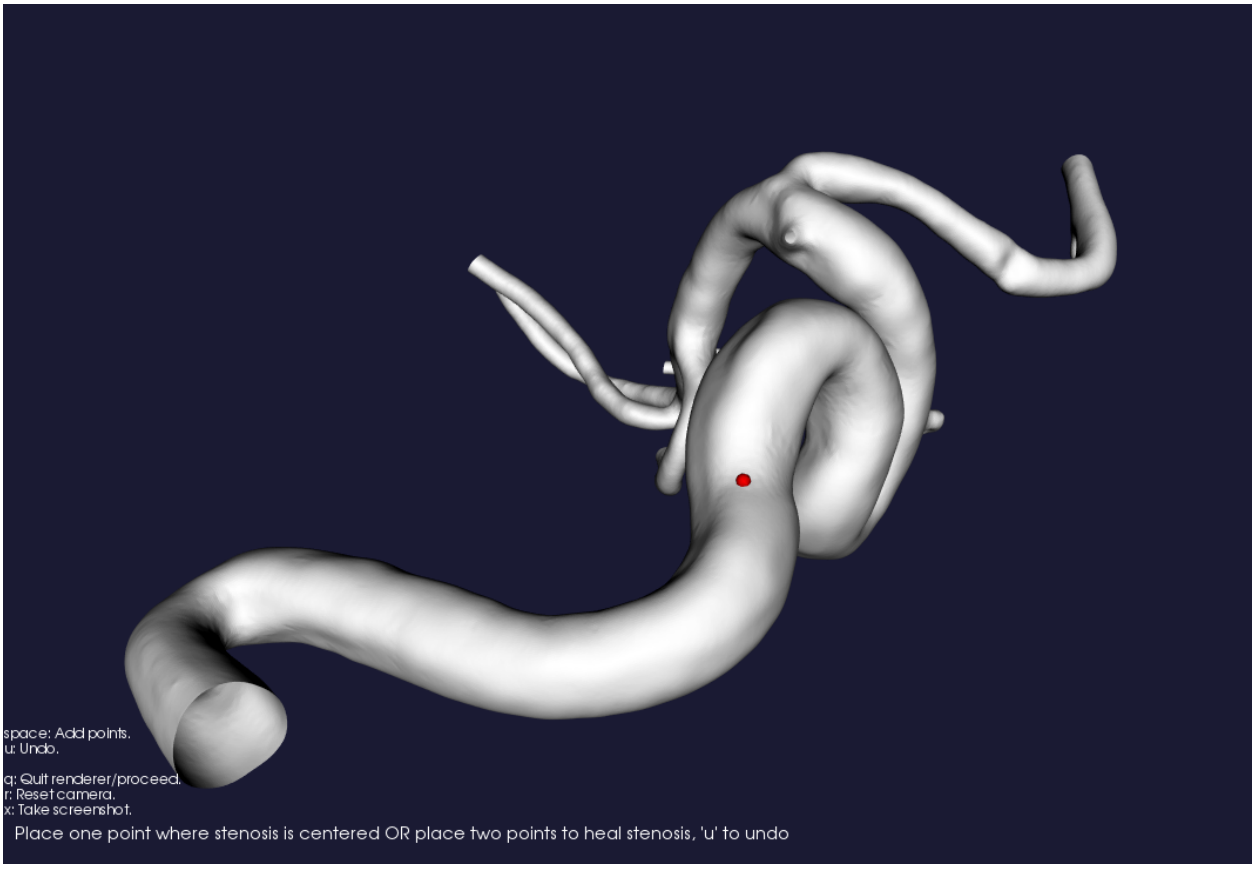

Fig. 9: Figure : Placing point where stenosis is centered.

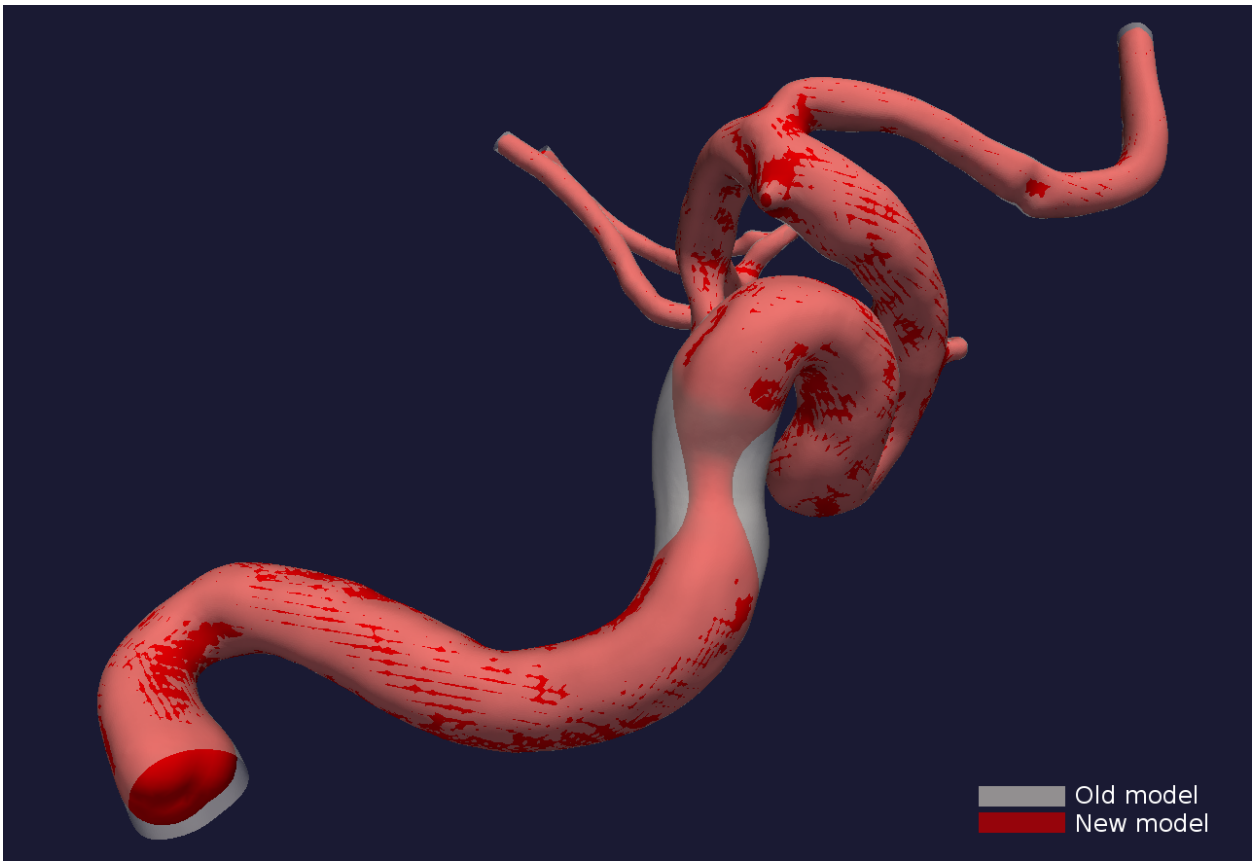

Fig. 10: Figure : Comparison of new and old model, with and without stenosis.

Similarly, removal of a stenosis is achieved by running the command:

```
python area_variations.py --dir_path [PATH_TO_CASES] --case [CASENAME] --smooth True -
↪-stenosis True
```

The command will cause a render window to appear, asking you to specify points on the surface which will now act as the boundaries of the stenosis.

Comment: Linear change in area between the two points.

This will trigger the following prompt, with a suggested point placement:

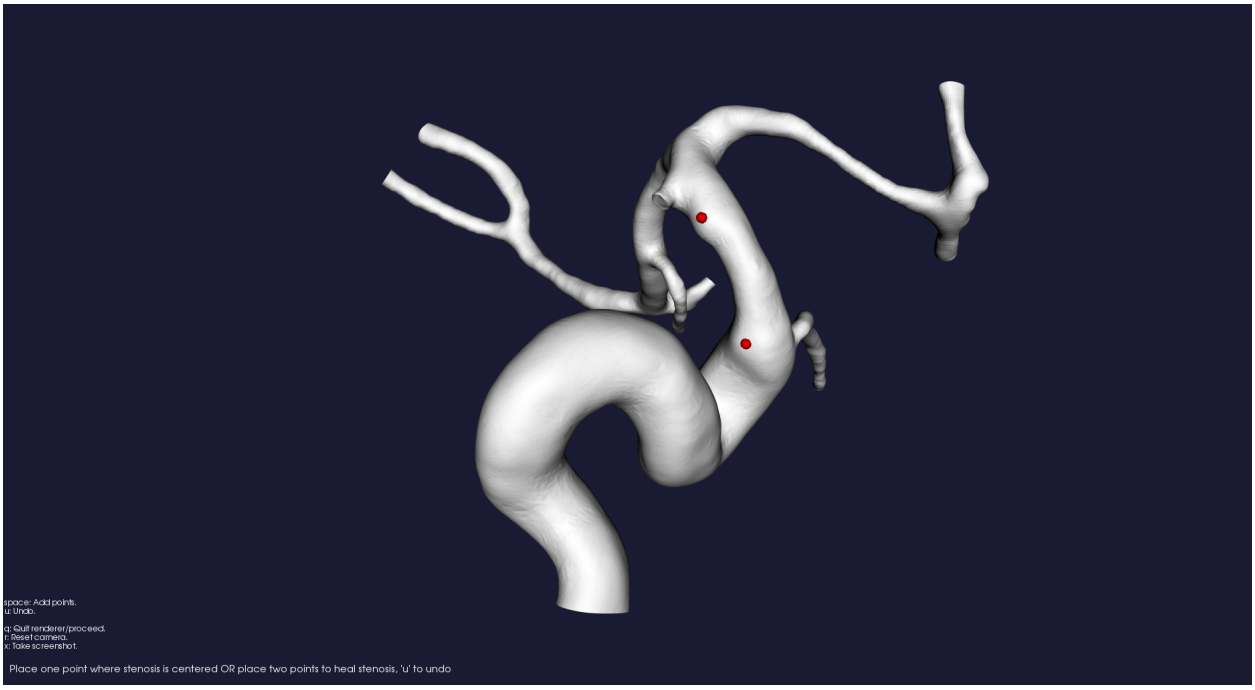

Fig. 11: Figure : Placing points to indentify the boundaries of the stenosis.

2.5.3 Curvature and angle variation of the anterior bend

Note:

Can be used for a general bend, but if used in ICA... Manipulation is initialized by selecting a segment of the vessel, bounded by two clipping points.

The two clipping points can be freely chosen along the centerline, but it is highly recommended to landmark the geometry in order to objectively segment the geometry, and use the resulting landmarking points as clipping points.

Adjusting curvature and angle in the anterior bend utilizes a common script: `move_siphon.py`. The script performs geometric manipulation of the anterior bend segment, as defined in the landmarking section. Adjusting the anterior bend relies only on two parameters, the compression/extension factors α and β. Alteration of the curvature or angle of the anterior bend is performed by specifying these factors in the script `automated_geometric_quantities.py` and `calculate_alpha_beta_values.py`. The pipeline for increasing or decreasing either curvature or the bend angle in the anterior bend is described below.

Alternatively the user may choose any arbitrary values for α and β.

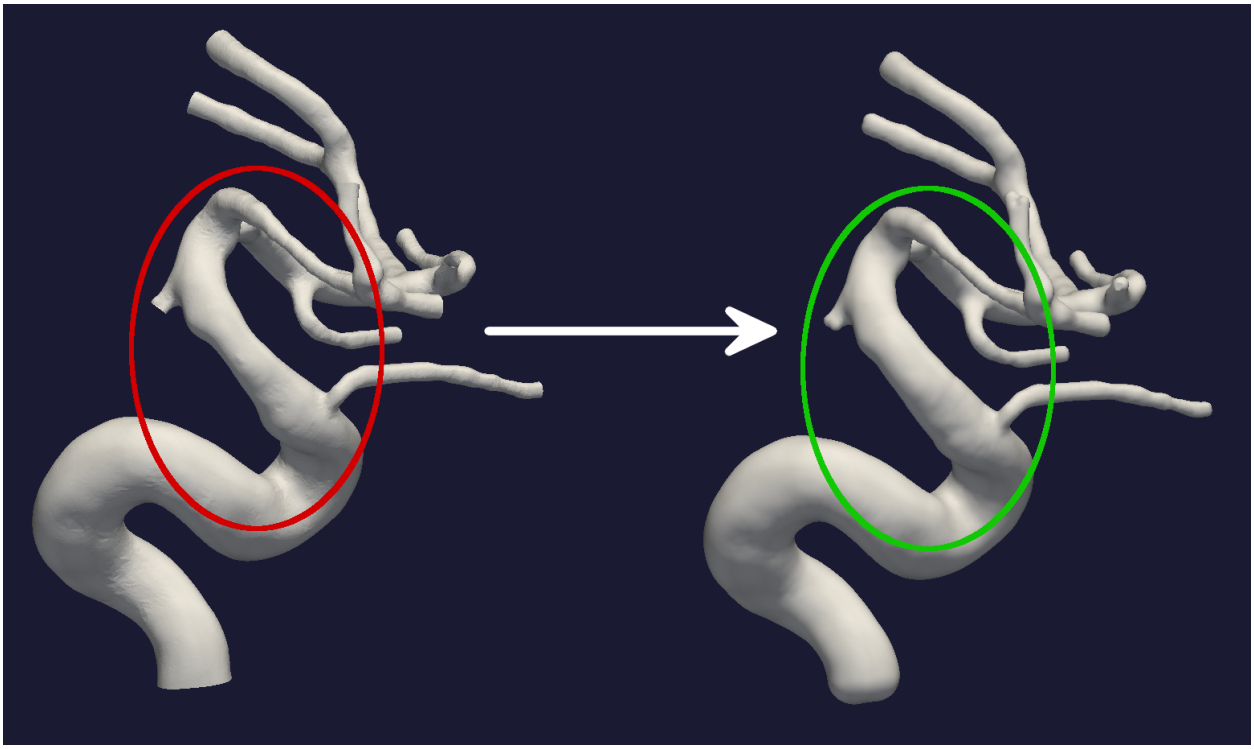

Fig. 12: Figure : Comparison of new and old model, with and without stenosis.

To perform geometric manipulation of the anterior bend, run the following command:

```
python move_siphon.py --dir_path [PATH_TO_CASES] --case [CASENAME] --alpha [ALPHA] --
↪beta [BETA]
```

In general, the compression / extension factors α and β determine the magnitude and direction in which the anterior bend is translated. The manipulation script allows movement in two directions:

- Vertical, determined by α
- Horizontal, determined by β

2.5.4 Curvature and torsion variation in the vessel

Selection of compression / extension factors

The compression / extension factors α and β determine the magnitude and direction in which the anterior bend is translated. Running the scripts `automated_geometric_quantities.py` and `calculate_alpha_beta_values.py` is required if the user is interested in reaching a spesific change in angle or curvature.

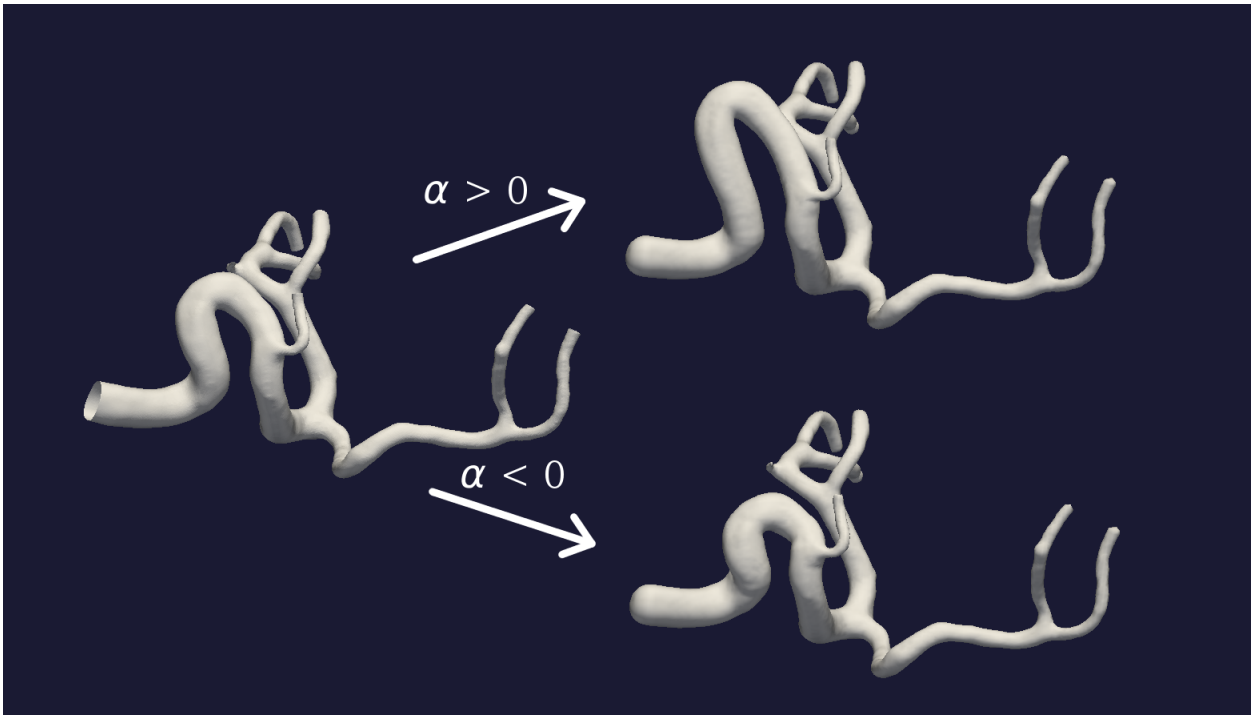

Fig. 13: Figure 6: Movement in the vertical direction, determined by α. FIXME: New model, old model.

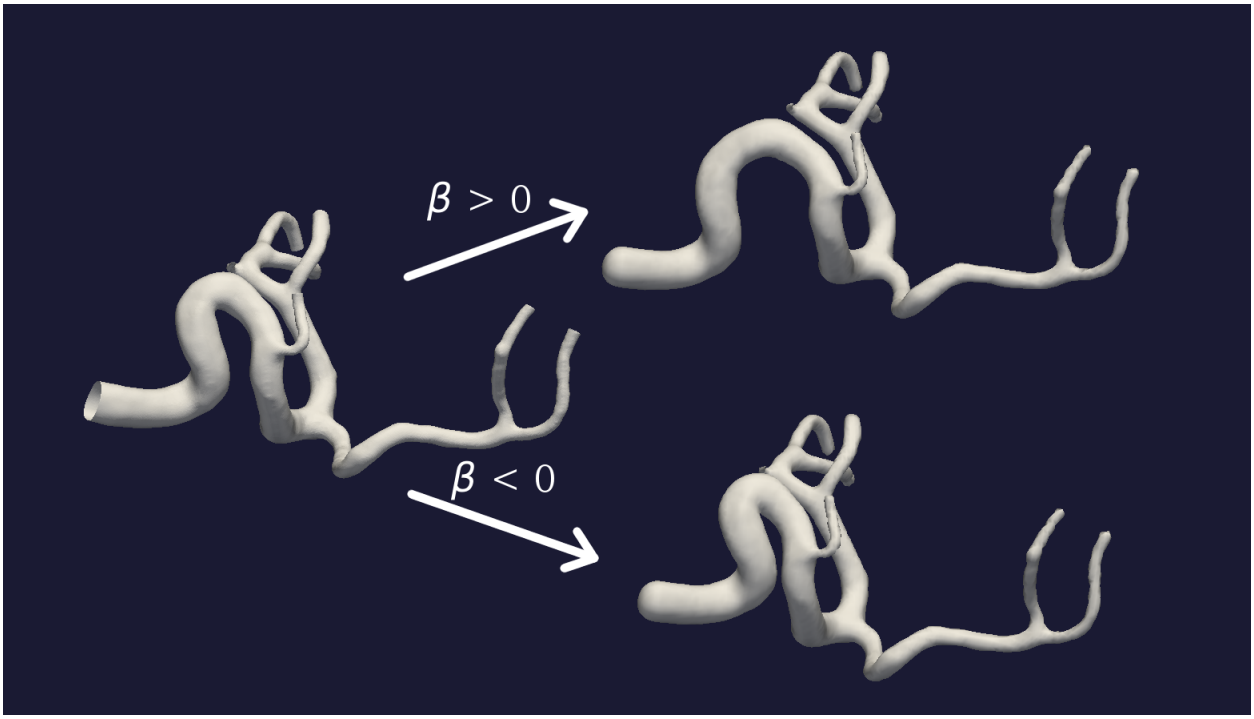

Fig. 14: Figure 7: Movement in the horizontal direction, determined by β. FIXME: New model, old model.

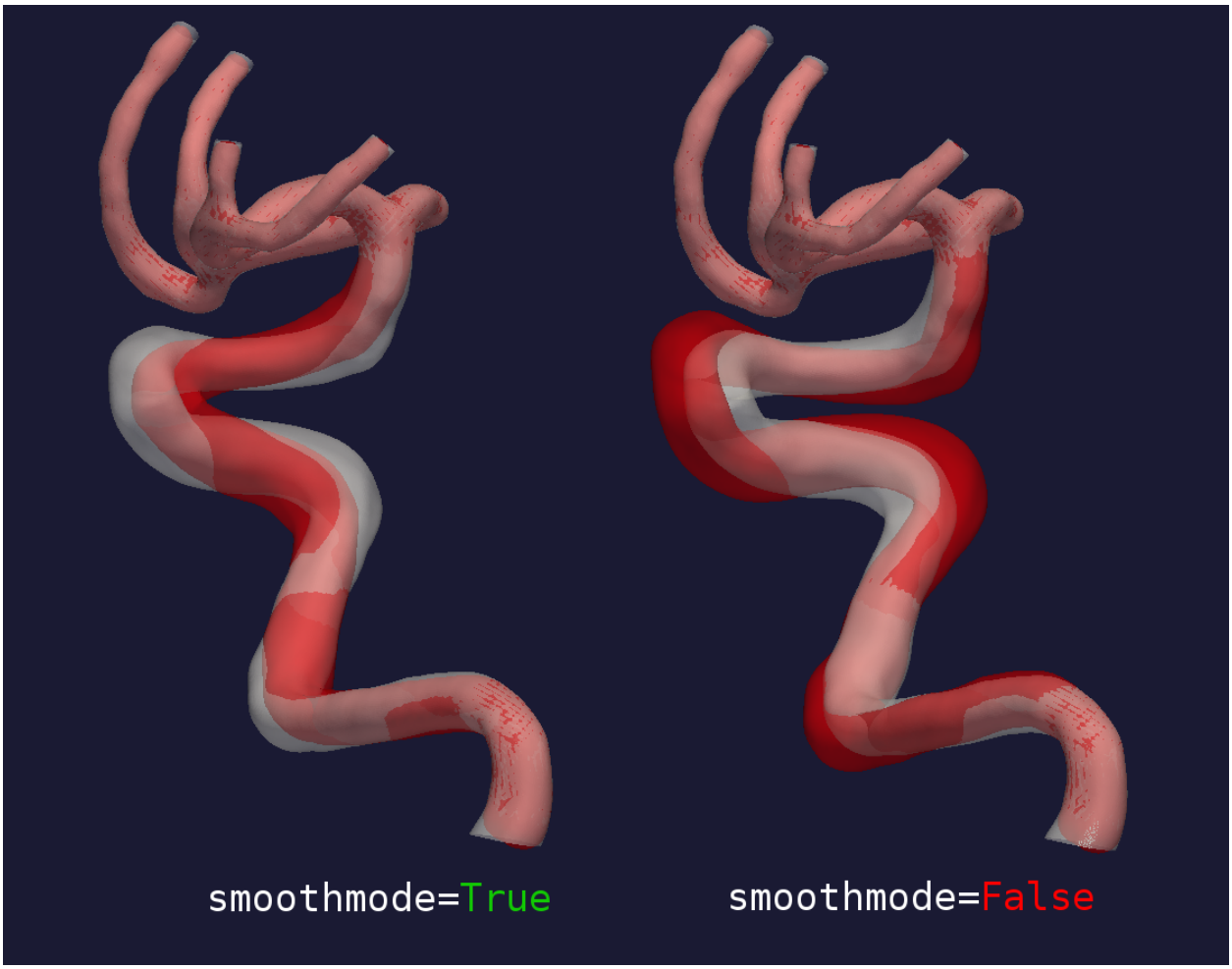

Fig. 15: Figure 7: Sharpened and smoothened version of the siphon.

CHAPTER

THREE

DOCUMENTATION FOR THE PYTHON SCRIPTS

3.1 Initialization

3.1.1 initialize_cases.py

3.2 Landmarking of the Carotid Siphon

3.2.1 automated_landmarking.py

Todo: Include Piccinelli

3.3 Manipulation of the Carotid Siphon

3.3.1 move_siphon.py

3.3.2 calculate_alpha_beta_values.py

3.4 Computing angle and curvature of the Carotid Siphon

3.4.1 automated_geometric_quantities.py

3.5 Manipulation of the ICA Bifurcation

3.5.1 move_branches.py

3.6 Miscellaneous

3.6.1 common.py

3.6.2 moveandmanipulatetools.py

Todo: Provide some text before all the docstrings?