Modelling of realistic Blood Vessel Geometry

Gottfried Wilhelm Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Mensch-Maschine-Kommunikation
Fachgebiet Graphische Datenverarbeitung

Bachelor Thesis of

Sergi Lázaro

Erstprüfer: Prof. Franz-Erich Wolter
Betreuer: Dr. Karl-Ingo Friese

Hannover, September 28, 2011

Contents

1 Introduction

1.1 Motivation

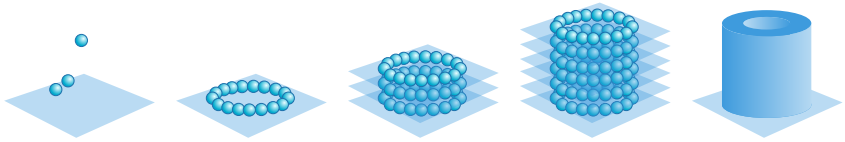


Figure 1: Overview of the 3D printing process for living tissue.

In the field of biological engineering, recent advances have made possible the printing of 3D living human tissue, with the help of laser technology (see figure 1). It consists of sticking cells together layer by layer, so as to build a general structure in the end. They voxelize a triangle mesh with their own software, and print the 3D matrix of cells in a similar way to current 3D printing. This is currently in development in the Laser Zentrum Hannover e.V., which this thesis is made in collaboration with. In their current setting, they can print the 3D living structure of human cells; however, it quickly dies given the lack of proper sustenance (i.e. blood).

Their objective now is to add the vascular system to the current structure, in order to be able to feed the cells and keep them alive, both during and after the creation process. For that purpose, they need a tridimensional model of the blood vessels, that must follow certain parameters in order to be useful.

1.2 Previous work

There are not many papers or projects on a similar topic to what we need to do. The closest existing work to what we need is by Xuemei and Huan in [7]. In this paper, they propose an algorithm to create visually convincing 2D blood vessels, using Lindenmayer Systems (see section 2.2 on page 7). While the basic concept is similar (using L-systems to mimic the branching patterns, see figure 2), the objective is totally different. We want useful and working blood vessels, obviously in three dimensions, with real-life sizes and proportions, et cetera.

In addition, we only need the low-level sections of the veins and arteries, not the high-level structure of the vascular system. Therefore, we need less human input

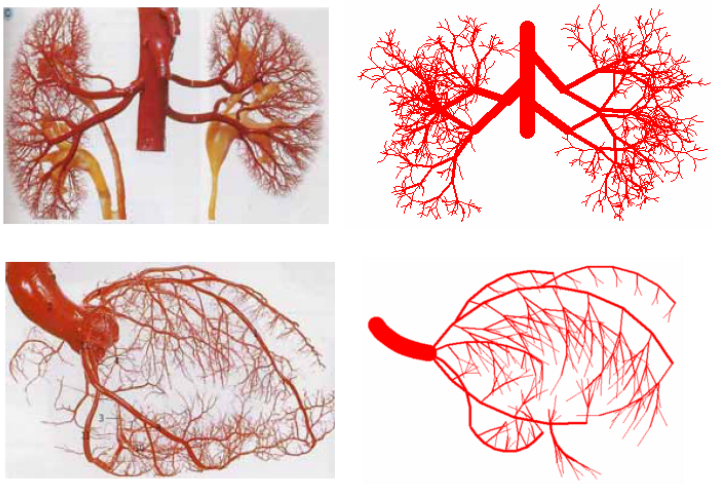


Figure 2: Results by the paper in [7]. On the left, real-life pictures of human blood vessels. On the right, their 2D results using their L-System.

for visually convincing structures, since it will be more uniform regardless of the organ it will be used for (see section 2.1 on page 7).

1.3 Objectives and Scope of Work

The objective of this thesis is to develop the program that will be used to create the tridimensional models for use with the laser 3D printing system. Only a 3D mesh is required so far, since there are existing tools to convert the mesh to volumetric model.

The requirements for the final blood vessel model are strong: as opposed to just a visually convincing model, where iterative improvements on the visual quality can be made until satisfactory, and intermediate stages may be good enough, in this project the resulting model has to satisfy some hard requirements:

- The vessels must be connectable at both ends to input and output valves.
- The walls must be thick enough to withstand the blood pressure.
- The bifurcations must have angles that are not too sharp to prevent the proper flow of blood.

- The proportions for input diameter and output diameters in each bifurcation must be such that the blood that goes in has enough room to come out.
- The maximum distance from any given cell to a blood vessel cannot exceed a certain threshold, so that all cells can get their sustenance.

Any of these requirements not being met would cause the model to be unusable.

All these requirements are too many to be properly addressed in a Bachelor Thesis. Therefore, the goal of this thesis will be to develop the system that can create a resulting 3D model, with the maximum amount of simplifications applied, and then iteratively improve the process in order to get as close as possible to the desired usable model. This way, a basic pipeline for the process will be created, and future work can improve and expand its capabilities in an iterative fashion until the result is the one needed.

1.4 Overview of the process

The process of creating the blood vessel geometry will start with the creation of the Medial Axis (section 2.4 on page 11) of the blood vessel geometry, being able to visualize it, with the possibility of tweaking the parameters in order to improve the structure. Then, the generation of the final mesh can be started, given the Medial Axis and the radius function, also defined during the previous tweaking.

2 Basic concepts

2.1 Blood Vessels

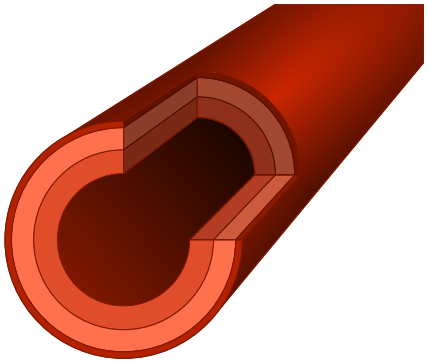


Figure 3: A section of a blood vessel. The inner space and the thickness of the wall are visible.

Human vascular systems have known properties both on the high level and on the low level. They differ greatly in size across the whole body: their diameter ranges from $25mm$ to $8\mu m$ [4].

On the high level, the arteries and veins are big enough to be visible to the naked eye, and display some structures that are specific to the part of the body they supply blood to. This structures are well known, and are followed with little variation in all healthy human beings.

On the lower levels, the shapes are not dictated as much by the zone they are in, but display instead some uniformity in the subdivisions, with also some amount of randomness. The patterns of subdivision are defined by local flow requirements, determined at the time the vessels are developed.

2.2 Lindenmayer systems

Lindenmayer systems, or L-systems, were introduced in 1968 by Lindenmayer [6]. They consist of rewriting rules that, when applied recursively, permit the creation of fractal-like or tree-like structures. Formally, they consist of a grammar

$$G = (\Sigma, \omega, P)$$

where

- Σ is the alphabet of symbols,
- ω is the initial symbol, and
- P is the set of substitution rules that determine the patterns.

There are several variations on this idea; one that proves useful for the later usage of an L-system as a geometrical representation, is the parametrical L-systems. This variation introduces numerical parameters to be added to each symbol, which can be, as in the next example, the length of a segment or the angle of bifurcation of a branch.

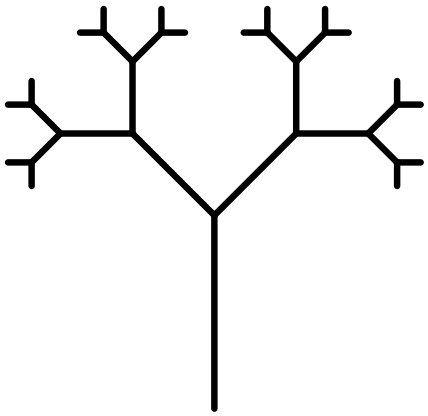


Figure 4: A tree generated by an L-system.

As an example, the parametrical L-system

$$G = (\Sigma, \omega, P)$$

with

- $\Sigma = \{a(\theta), s(l), t(l), [,]\}$,
- $\omega = t(1)$, and
- $P = t(l) \rightarrow s(.6l)[a(-45)t(.4l)][a(45)t(.4l)]$

would create, after four steps, a tree-like structure like Figure 4. In this system, the alphabet consists of three parametrical symbols and two simple symbols:

- $a(\theta)$: make a turn of θ degrees clockwise,
- $s(l)$: create a middle segment of length l (these segments will not be modified further),
- $t(l)$: create a leaf (terminal) segment of length l (these segments will be subdivided in further steps),
- [: start a new branch from the end of the previous middle segment,
-]: end of the branch that was started most recently;

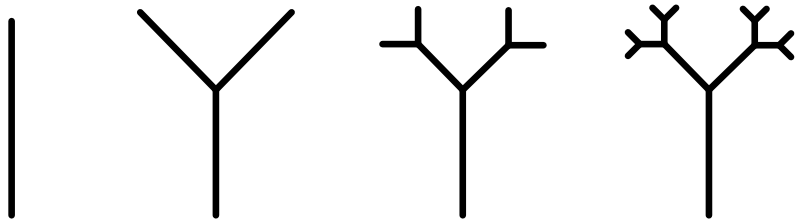


Figure 5: Graphical representation of the subdivision rule.

and the production rule shortens each leaf branch, and adds two smaller leaf branches to its end, separated 45 degrees from the original one, one at each side. The subsitution rule could be graphically represented as in Figure 5. These are the first four steps of the Lindenmayer System.

At the first step, we would have

$$t(1)$$

which corresponds to just one straight line of length 1. At the second step, we apply the substitution rule to the only segment that we have, yielding

$$s(.6)[a(-45)t(.4)][a(45)t(.4)]$$

which graphically would match the second image in Figure 5. First we have a segment of length .6, with two branches of length .4, separated at 45° from the main branch. The main branch is represented with the parametrical symbol $s()$, while the other two branches are $t()$. That means that the main branch will not be replaced with other branches in further steps, and the other two will.

The third step starts to be quite difficult to understand in text form; as we can see, the exponential growth that comes with L-Systems is apparent not only with the graphical trees that can be created with it, but also with the length of the words:

$$s(.6)[a(-45)s(.24)[a(-45)t(.16)][a(45)t(.16)]][a(45)s(.24)[a(-45)t(.16)][a(45)t(.16)]]$$

Other parameters that could be included in the substitution rule could be the diameter of each segment (either constant or varying within the segment), a second angle (torsion) in order to have the final model be truly tridimensional and not only lie in a plane, etc.

In order to transform the long strings of words to their graphical form, the well known method of Turtle Graphics is used.

2.3 Turtle Graphics

Turtle Graphics is a method for drawing vector graphics by controlling a cursor in a 2D plane. The most primitive turtle could be $T = (p, \vec{d})$, where

- p is the current position
- $\vec{d}$ is the direction vector

A basic set of the commands to operate it could be:

- `line distance`: draw a line of length `distance`, from p in the direction $\vec{d}$.
- `turn angle`: rotate $\vec{d}$ for `angle` degrees.
- `save turtle`: save the current position and direction to the turtle named `turtle`.
- `load turtle`: set the current position and direction to the previously saved turtle named `turtle`.

In the previous example of L-system, these commands map exactly to the terminal symbols in the system: $a(\theta)$ is `turn angle`, $s(l)$ is `line distance`, and $[,]$ would be `save turtle` and `load turtle`, respectively.

As an example, the sequence of Turtle actions that would generate the tree corresponding to the second step of the L-system example in page 9 would be:

```
line 0.6
save turtle
turn -45
line 0.4
load turtle
turn 45
line 0.4
```

For other, more complex L-systems, with different parameters, in 3D, etc., variations on the Turtle Graphics will have to be made, as will be seen in section 4.1.2 in page 16.

2.4 Medial Axis

The concept of Medial Axis (MA) was introduced in 1967 by Blum [2]. The MA of an object (a closed set in 2D or 3D) is the set of all points inside the object that have two or more closest ponts to the boundary. Explained in a different way, it is the set of all centers of maximal spheres (in 3D, or circles in 2D) contained in the object. A sphere in the object is maximal if there's no other sphere (also contained in the object) that contains it.

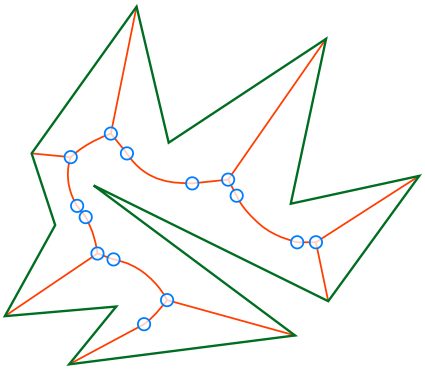


Figure 6: A simple polygon (green), its Medial Axis (red), and nodes of the tree (blue).

Informally, the Medial Axis of an object can be thought of as its skeleton: it will consist (mainly) of a tree-like structure of lines that go through the "middle" of its "limbs". An example can be seen in Figure 6.

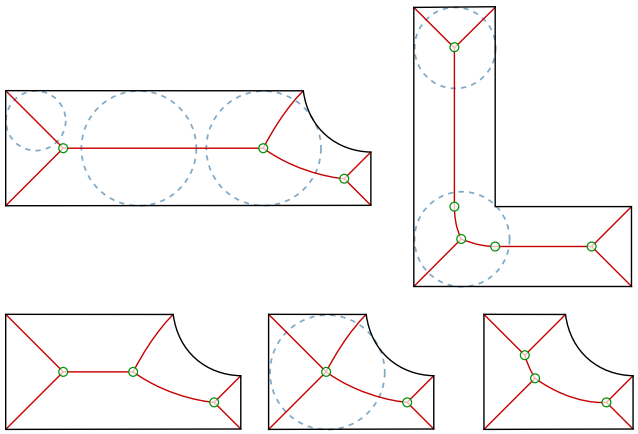
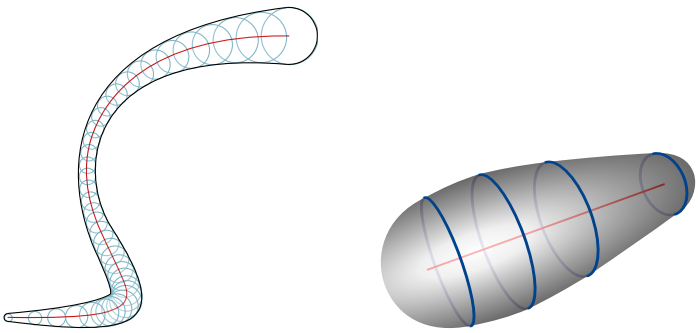


Figure 7: Different shapes (black), their Medial Axes (red), their nodes (green), and some of the maximal circumferences (blue)

Figure 7 shows two examples on top, plus three variations of the first example on the bottom row. With these three variations it can be seen that the topology of the Medial Axis as a graph changes when two nodes join. Two nodes joining means that the maximal circumference centered there shares their tangential points to the shape.



(a) 2D example. (b) 3D example.

Figure 8: Black curve (a) and gray surface (b), created by applying the radius function to the red axes, and some example circumferences in blue.

Additionally, the Medial Axis can have a radius function associated with it: for each point in the axis, it defines the diameter of the "limb" of the object at that point. This function together with the MA is called the Medial Axis Transform,

since by applying the radius function to the Medial Axis the original object can be reconstructed, in a process called the back-transformation of the Medial Axis. An example can be seen in Figure 8.

In our project, we will interpret the output of our L-System as the Medial Axis of the blood vessel and then we apply the back-transformation to it in order to get the blood vessel shape itself.

3 Literature

3.1 Similar work

As seen in section 1.2, the other existing previous work by Xuemei and Huan in [7] shares some similarities with the objective of this thesis, but also has some important differences. Expanding on previous work in the same subject by Zamir in [11], it focuses mainly on visually appealing 2D images of blood vessels. Obviously we want 3D models, with wall thickness, and we don't focus on visual similarity: we want something usable; whether it looks regular or not is irrelevant.

3.2 Lindenmayer Systems

The first definition of Lindenmayer Systems was introduced by the own Lindenmayer in 1968 [6], it has been used in numerous publications since then.

In 1991, Lindenmayer himself and Prusinkiewicz explain in depth the usage of L-systems and Turtle Graphics for generation of plants [8], which is applicable to our blood vessel structure.

A very detailed look on Turtle Graphics alone is available in the book *Turtle geometry* [1].

3.3 Medial Axis

Blum introduced the mathematical concept in a paper in 1967 [2]. Its implementation, and applications, both for the 2D and the 3D case, has been worked on since then.

Wolter in [9] and with Friese in [10] give overviews of the modelling of solids using the Medial Axis Transform, computation of the MA given the solid, and reconstruction of the solid given the MA and the radius function.

In the Diploma Thesis of Böttcher [3], improvements of the modelling using Medial Axis Transform are introduced, as well as the construction of shapes given the input data for the program. The computation of the surface given the MA is key to our thesis, and is covered in depth in this work.

4 Tree structure

There are two main steps for creating the 3D mesh.

In the first step for creating the mesh, the general structure of the tree needs to be created (section 4.1). That is, to know where in space each node of the tree is located, which diameter and wall thickness it has, and its connectivity. In order to do this, the parameters for the generation of the tree have to be calculated given the user input data (section 4.2).

After that, given the information of the structure (Medial Axis) and thickness at each point (radius function), the 3D mesh has to be generated (section 5).

4.1 Building the tree structure

In order to create the structure of the tree, an L-system (section 4.1.1) is used with a simple set of parameters, and the actual position of each node is found by "drawing" with a 3D version of Turtle Graphics (section 4.1.2).

The L-system just produces one part of the whole vessel structure, since the end result is not exactly a tree (it has cycles). As it can be seen in Figure 10 on page 18, the tree consists of start and end vertical segments, middle capillaries, and two symmetrical (actual) "middle" trees (in the figure, marked with h_m). Therefore, only one of those trees is generated, and the other one is created by reflecting the original one through the XY-plane that would cut the whole structure in two halves.

4.1.1 Our L-system to generate the "middle" tree

The L-system that we will use for the generation of the tree will be

$$G = (\Sigma, \omega, P)$$

with

- $\Sigma = \{d(n), b(n, \theta), s(L), r(\phi), y(\theta), [,]\}$
- $\omega = d(0)$

- $P = \{d(n), b(n, \theta)\}$
- $d(n) \rightarrow [b(n, \theta_n)][b(n, -\theta_n)]$
- $b(n, \theta) \rightarrow y(\theta)s(L_n)y(-\theta)r(\phi_n)d(n + 1)$

where the parameters mean:

- $d(n)$: "dummy" symbol to create the recursion, it will just be substituted for level n in the tree.
- $b(n, \theta)$: branch in level n, with bifurcation angle θ.
- $s(L)$: create a segment of length L.
- $r(\phi)$: do a roll-turn of angle ϕ, for torsion.
- $y(\theta)$: do a yaw-turn of angle θ, for bifurcations.
- $[,]$: push-pop symbols to save-restore the position, in order to define both branches at the same level.

With this L-system definition, there is only one bifurcation angle and torsion per level (θ_n and ϕ_n, respectively). This can be changed for variations on the tree generation, like randomizing the torsion angle to add a more organic feel, but for our application it's sufficient.

4.1.2 3D Turtle graphics

The actions of the helper 3D Turtle graphics that we need are defined by the L-system that it will be used with. That is, for each terminal symbol in the system, we will need an action for the Turtle. Therefore, we will need the standard `line distance`, `save turtle` and `load turtle`, as seen in Section 2.3.

For the angles, we need something more than the 2D `turn angle`. In addition, since we need 3D movement and orientation, a single position point p and direction vector $\vec{d}$ will not be enough. Our 3D Turtle T will be (see figure 9):

$$T = (p, \vec{d}, \vec{u})$$

where

- p is the current position

- $\vec{d}$ is the direction vector
- $\vec{u}$ is the up-vector, perpendicular to $\vec{d}$

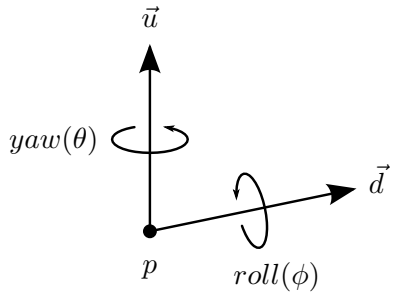


Figure 9: The 3D Turtle vectors and their rotations.

With this, $\vec{d}$ is still used for drawing straight lines, and the new $\vec{u}$ is used only for the rotations. We have two new `rotate` actions: `rotate_yaw angle` rotates around $\vec{u}$, and `rotate_roll angle` rotates around $\vec{d}$, by modifying these two vectors.

4.2 Calculating L-system parameters given higher-level parameters

The input parameters for creating the vessel structure are not the ones that the Lindenmayer System uses to create the tree. As seen in section 4.1.1, our L-system requires a length L_n for each level, plus a bifurcation angle θ_n and a torsion angle ϕ_n. But giving all of these parameters would be tedious, and we need some control on the total width and height of the tree, as well as some other higher-level requirements. Therefore, some calculations have to be made in order to derive the parameters needed for the L-system.

4.2.1 Input parameters for the program

The parameters (see Figure 10) are:

- bounding box dimensions: w_b, h
- for start vessel s and capillaries c,
  - length of the segment: L_s, L_c

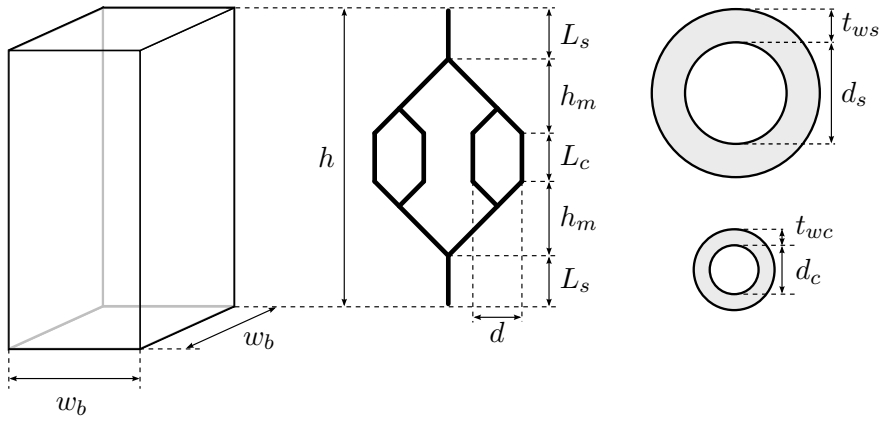


Figure 10: The parameters for the generation of the tree.

  - diameter: d_s, d_c
  - wall thickness: t_{ws}, t_{wc}
- capillary spacing d

One of the requirements for the laser-printed structure is that the distribution of the capillaries at the end has to be uniform, and cover a given area. This requires some mathematics, that are easier solved when considering 2D trees.

4.2.2 Solving for 2D trees

A binary tree divides the space in halves, recursively (see Figure 11), when the lengths of the segments L_n and the bifurcation angles θ_n follow some properties:

$$\theta_n = \theta_{n+1} = \theta \quad \forall n$$

$$L_n = 2L_{n+1} \quad \forall n$$

Therefore, to calculate the angle θ, we need the height of the "middle tree" h_m and the covered width c (see Figure 12), which are calculated from some of the parameters:

$$h_m := \frac{h - 2L_s - L_c}{2}$$

$$c := w_b - 2m, \quad where$$

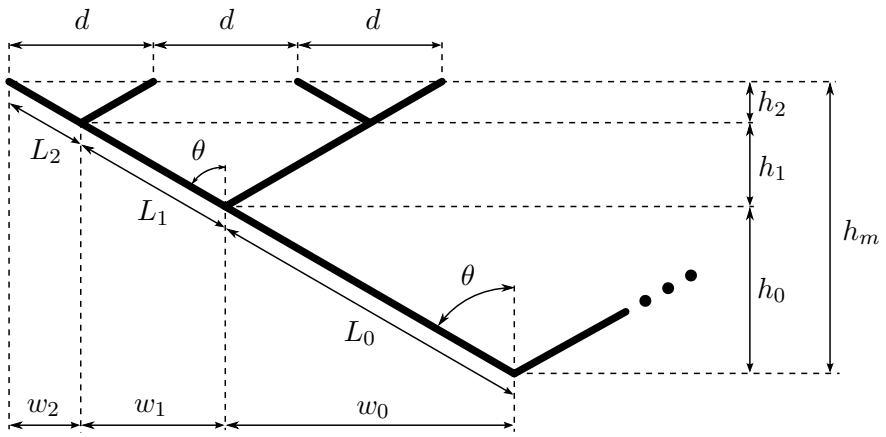


Figure 11: The trigonometrical properties of the branches of the tree at various levels.

$$m := \frac{d}{2}$$

Given the capillary spacing d and the covered width c, we can calculate the number of levels of the tree N. Since $N \in \mathbb{N}$, we have to use the minimum number of levels so that the distance between end points is less than c:

$$N := \lceil log_2(\frac{c}{d}) \rceil$$

In order to calculate the lengths L_n, we need the widths w_n, which can be calculated by dividing the width by two, recursively. But since this would be a geometric series, it would never reach the covered width c. Therefore, we need to calculate a new width w, slightly larger than c, so that c is reached after N levels of subdivision.

As it can be seen in Figure 13, $w_0 = \frac{1}{4}w$. Going forward, we can see that the general formula is

$$w_n = \frac{1}{2^{n+2}}w, \quad \forall n < N$$

and also

$$c = 2\sum_{n=0}^{N-1} w_n$$

but we can also see that

$$c = w - 2w_{N-1} = w - \frac{1}{2^N}w = w(1 - \frac{1}{2^N}) = w\frac{2^N - 1}{2^N}$$

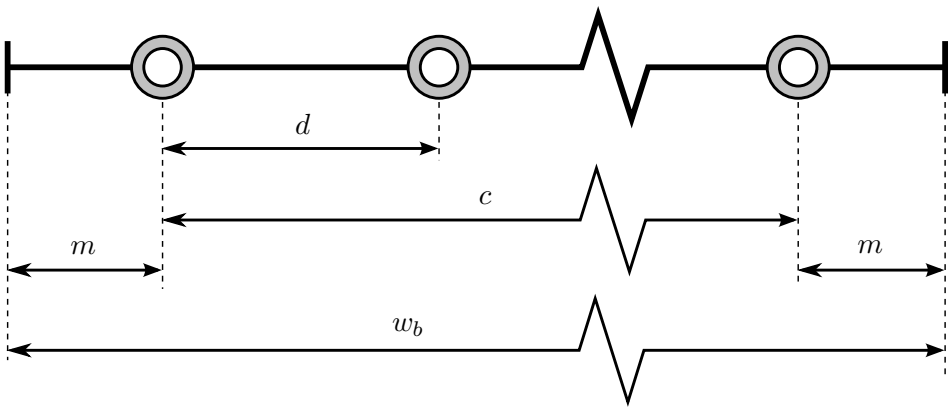


Figure 12: The sizes and distances at the capillary level.

therefore, we can calculate w now:

$$w := c\frac{2^N}{2^N - 1}$$

With all this information, now we have to calculate the angle θ and the lengths L_n, which are the only parameters needed for the L-system.

$$\theta = |\arctan(\frac{c}{2h_m})|$$

$$L_n = \frac{w_n}{\sin(\theta)}$$

The number of segments in the tree and the number of nodes can be calculated like so (see Figure 14):

$$N_{segments} = 3 + 2N$$

$$N_{nodes} = N_{segments} + 1$$

Additionally, we will have a diameter and wall thickness for each node of the tree (d_n and t_n, respectively). Since we only have the values at the start and capillaries (t_{ws}, d_s, t_{wc}, d_c), the rest of the values for the "middle tree" will have to be interpolated. The values can be created just for the first half of the tree, since the tree is symmetrical:

$$\forall n \in [\frac{N_{nodes}}{2}, N - 1] :$$

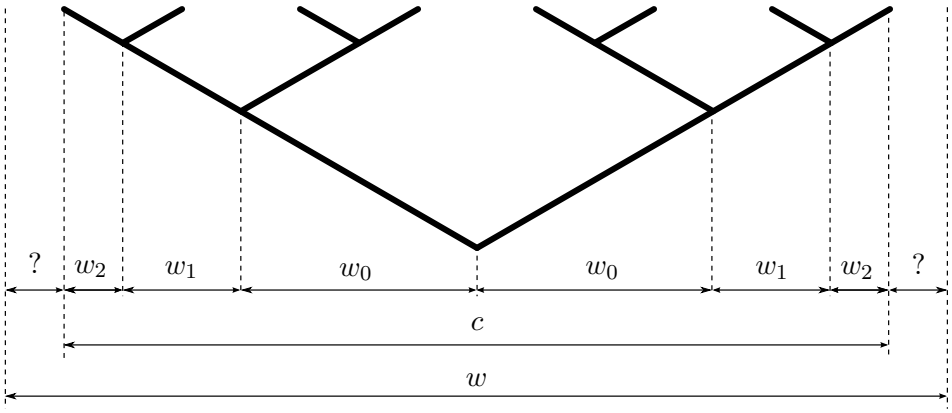


Figure 13: The recursive subdivision of the tree, by halving widths, when $N = 3$.

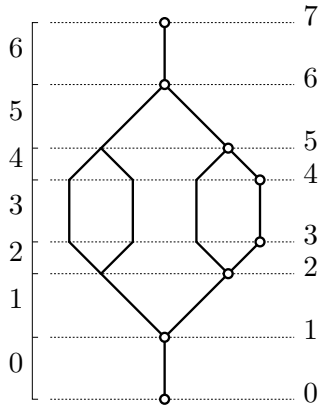


Figure 14: The numbering of segments (left of tree) and nodes (right of tree), when $N = 2$.

$$t_n := t_{N_{nodes}-n-1}$$

$$w_n := w_{N_{nodes}-n-1}$$

In order to make it with a constant rate along the way, it has to be interpolated in relation to the accumulated length from the beginning of the tree ($al(n)$):

$$al(n) = \begin{cases} 0 & n = 0 \\ al(n-1) + L_{n-1} & 0 < n < N_{nodes} \end{cases}$$

$$al_{total} := al(\frac{N_{nodes}}{2} - 1)$$

$$i_n := \frac{al(n)}{al_{total}}$$
$$d_n := lerp(i_n, d_s, d_c)$$
$$t_n := lerp(i_n, t_{ws}, t_{wc})$$

where

$$lerp(i, start, end) = start + i \cdot (end - start)$$

4.2.3 Using the parameters for the 3D tree

All the calculations so far have been made assuming the tree is 2D. But for our application, the tree is actually 3D. For that, we will use a "fake" 2D tree to get its parameters, which will be adapted to use for the 3D tree.

First, we will calculate θ for the 2D tree of half the height h_m, with all the other parameters being the same. The number of levels N for the 3D tree will be twice the value of the 2D tree.

The 3D tree will be formed by adding a torsion angle ϕ of 90º, so that the 3D turtle (section 4.1.2) does a roll-turn of ϕ as well as the yaw-turn of θ. This way, it's as if there is two half-trees growing in the XZ and the YZ planes, interleaving levels with each other.

All L_n, d_n and t_n values are calculated for the assembled 3D tree, by following the same rules as with the 2D trees.

4.3 Summary

The process for calculating the tree structure fill finally be:

Algorithm 4.1 Overview for calculating the tree structure

calculate N and ϕ given the input parameters
calculate w given c and N
for $n < N$ **do**
  calculate w_n given w and n
  calculate L_n given w_n and ϕ
end for
generate tree with the L-System given N, ϕ and $\{L_n \mid n < N\}$
for all $node \in nodes$ **do**
  calculate d_n by interpolating from d_s and d_c
  calculate t_n by interpolating from t_{ws} and t_{wc}
end for

5 Mesh Generation

Once we have the needed information of the structure of the tree (Medial Axis) and thickness at each point (Radius Function), the 3D mesh has to be generated. This consists in applying the back-transformation of the Medial Axis with the Radius Function.

One important observation has to be had in mind for all this process: as seen in section 2.1, the sizes of the vessels vary greatly in diameter, which means that approaches based on voxelization will have some shorcomings, and any resulting mesh will have big differences in triangle size, unless this is specifically addressed.

5.1 The back-transformation of the Medial Axis

The back-transformation of the Medial Axis is an open problem, that can be solved with different approaches, depending on the specific characteristics of the data to transform.

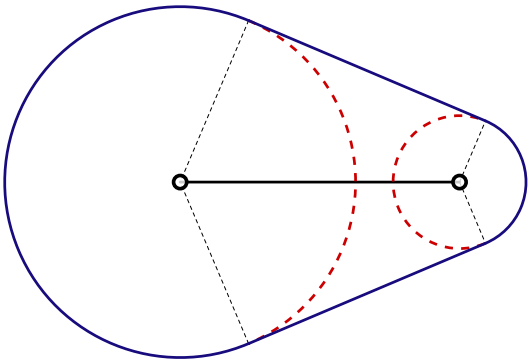


Figure 15: A 2D Capped Truncated Cone (CTC) in blue. The maximal circles at start and end in red, the corresponding medial axis in black.

Our particular case deals with linear (as opposed to planar) Medial Axes, composed of a 3D graph of several linear segments connected to each other at their ends by nodes. The values of the Radius Function (RF) are known at each node, and the RF for each segment consists of the linear interpolation of the values at both ends. Therefore, for the simplest case of only one Medial Axis segment, the back-transformation would look like figure 15.

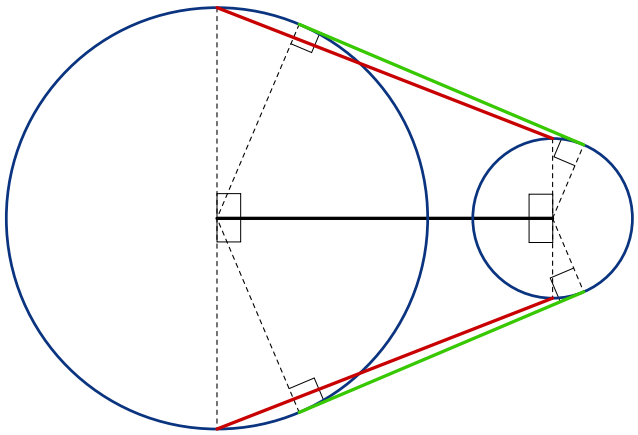


Figure 16: The proper circle tangency (green), compared with the naive, incorrect way (red).

The first, intuitive approach at creating the surface would be to "sweep" a circle of varying size along the medial axis, with radius equal to the value of the Radius Function at that point of the Medial Axis. This poses two problems: The first one is that there would obviously be intersections at the bifurcations of the MA, which should be somehow resolved. The second one is that it would yield incorrect results, caused by a common misconception on the nature of circle to circle (or sphere to sphere) tangency (see Figure 16). The tangencies between circles are not found on the points perpendicular to the center-to-center segment (pictured in red in the figure): the proper tangency is found at a certain slope, easily found using trigonometry (pictured in green in the figure).

Figure 17 shows the appearance of a bifurcation in 2D. It can be seen that the shape (in blue) consists of parts of the circles at the ends, plus the tangent lines between the circles of each pair of adjacent nodes. The intersections (in green) have to be computed, but that is an easy calculation.

Upon further inspection, it becomes apparent that resolving the shape at the intersections is the hard problem (see Figure 17), and even harder in 3D.

Figure 17 also raises an interesting fact about the Medial Axis transform: You can have a Medial Axis and a Radius Function that, when used to get a surface with the back-transformation, yield a surface that has a different MA+RF pair. Depending on the particular case, one or both of them will be different.

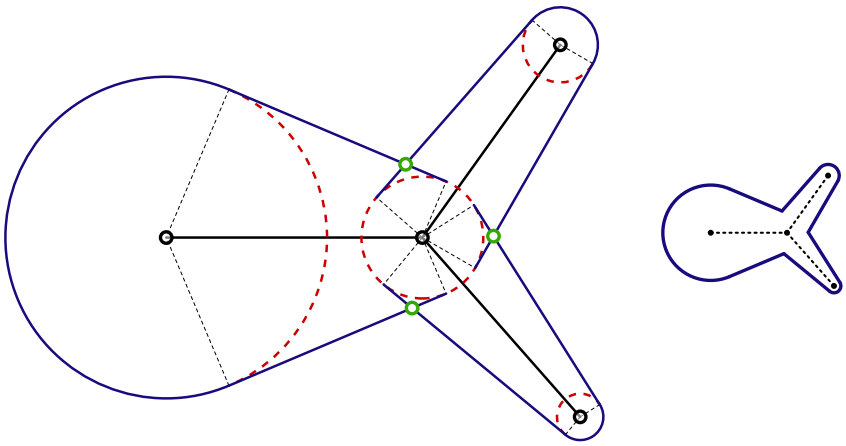


Figure 17: A Medial Axis bifurcation in 2D. The maximal circles at each node are in red, a first approximation to the desired shape in blue, and the intersection points in green. On the right, the final shape from the Medial Axis, in blue.

It is made more explicit in figure 18, where the RF at the bifurcation gives the red circumference, and the tangency lines in blue show that the shape is bigger at that point: the Radius Function at that point should be the green circumference, that passes through the intersections of the tangency lines. This is the Radius Function that would be calculated from the shape.

Another way to construct the final shape for the bifurcation in figure 17 is to compute the union of the basic shapes (figure 15), as seen in section 5.2.

In order to calculate the union of the shapes, the method of Constructive Solid Geometry is used, which can be applied to 2D and 3D shapes.

5.2 Constructive Solid Geometry

Constructive Solid Geometry (CSG) consists of a set of boolean operations on basic solids that, applied in a sequence, allows the construction of more complex shapes. The operations typically consist of Union ($\cup$), Intersection ($\cap$) and Difference ($-$), and are applied to pairs of solids to get a new solid (see figure 19).

In our case, applying the union of all the CTCs of each segment would yield the final surface. In broad terms, we will calculate two solids that will represent the

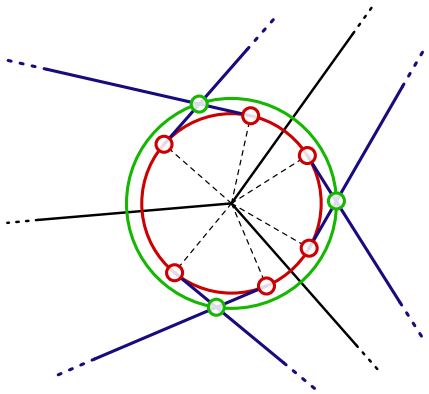


Figure 18: The inconsistency in radius value at the bifurcation point. Medial Axis in black, tangent lines in blue, red circle of original radius (with tangency points also in red), and green circle of proper radius (with intersection points also in green).

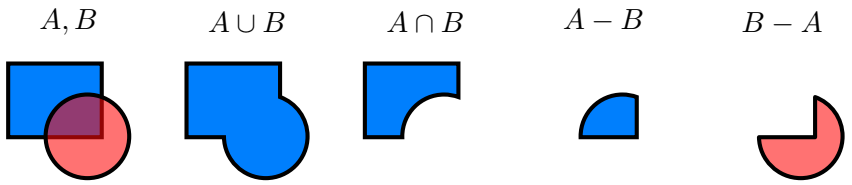


Figure 19: Visual example for the usage of CSG in 2D, based on the shapes A and B, applying the operations Union ($\cup$), Intersection ($\cap$) and Difference ($-$).

whole tree, one thinner than the other, so that the first one represents the outer side of the vessels and the second one the inner side. For the complete vessel structure, the sequence of operations to apply would be as follows (illustrated in figure 20):

1. For both thicknesses:
   a) For both halves of the tree, calculate the union of all its segments.
   b) Unite both halves, as well as with the capillaries.
2. Subtract the thinner solid to the thicker solid.
3. Intersect the resulting solid with a box, in order to open the ends.

In order to perform CSG operations with our code, the Java3D library J3DBool (`http://unbboolean.sourceforge.net/`) has been used. This library uses `Solid`

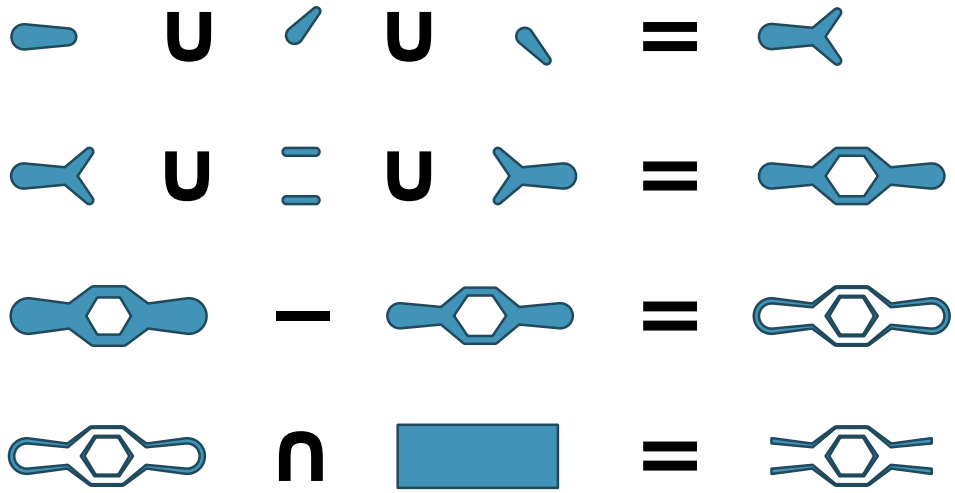


Figure 20: The stages of the CSG process to get the final mesh, for a simple tree (in 2D).

objects that consist of *b-rep* `IndexedTriangleArray` meshes, and applies boolean operations to them, yielding further `Solids`; it implements an algorithm described in 1986 by Laidlaw et al [5]. Using this library solves the mesh generation part, by following the steps in the previous enumeration.

However, this library proved to be a problem in the long run, since it only works properly under certain `Solid` sizes, running into infinite loops otherwise, and with an unmanageable time growth in terms of number of segments of the Medial Axis tree.

5.2.1 Splitting into nodes

In order to fix the problems with the `Solid` sizes and the growth of time, one solution was devised. Each CTC is split into two Half CTCs (HCTCs), as illustrated in figure 21. Therefore, each HCTC belongs to only one node, and the joining can be done per-node, avoiding the time growth in the CSG algorithm, and then stitched together, as seen in section 5.2.2. Also, the HCTC group to be joined can be scaled to a manageable size that will avoid the infinite loop problem, as seen in section 5.2.3.

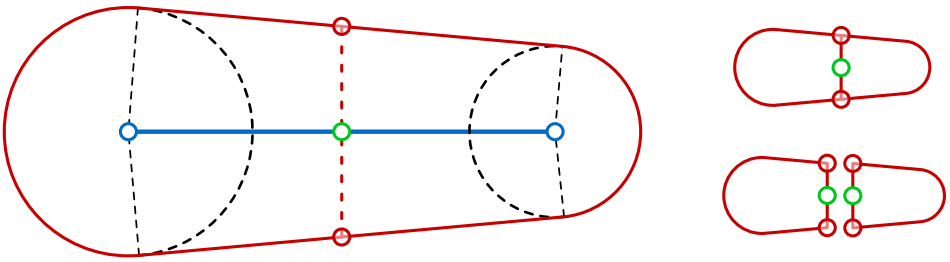


Figure 21: Splitting a Capped Truncated Cone (CTC) into two Half CTCs (HCTCs). In green, the center of the straight cap, important for stitching them back together.

5.2.2 Scaling and stitching together

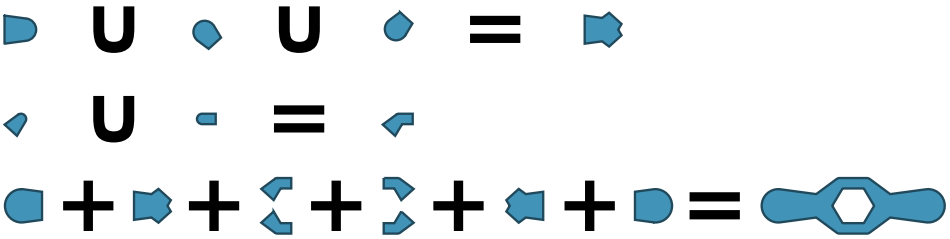


Figure 22: The updated stages of the CSG process to get the final mesh, when the joining is done per-node. An additional operation symbol (+) added to represent simple merging of triangles, as opposed to binary union (∪). The rest of the stages remain as in figure 20.

The process now will consist of joining each node separately (avoiding the time growth, since the CSG operations will be applied independently for groups of up to three shapes), and then "stitching" them together, as seen in figure 22.

In order to stitch them together, we need to remove all faces connected with the center of the straight cap (in green in figure 23), and connect all vertices around it (in red in the figure). For this, we need to keep count of which are the `IndexedTriangleArray` indices of the vertices to stitch together, as well as the center of the straight cap.

The scaling and joining process of each node consists of three steps (see figure 24):

1. Scale the node up: since we just modify the coordinates of each vertex, its indices remain the same.

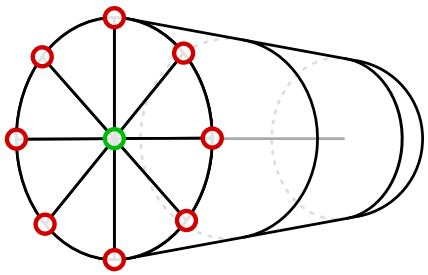


Figure 23: A Half CTC (HCTC) in 3D. The center of the straight cap is in green, and the vertices to be stitched are in red.

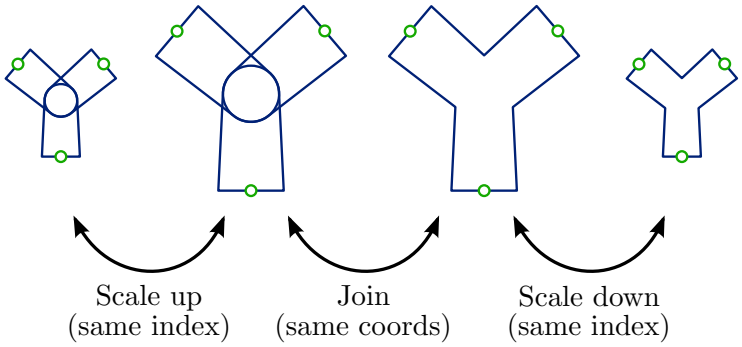


Figure 24: Steps to follow when joining and scaling each node, in order to preserve vertices that shouldn't have changed (in green)

2. Join them together: using CSG as seen before, the coordinates for the vertices in the straight cap won't be changed, since they are not part of the bifurcation *per se*. However, we can't guarantee that the indices are unchanged as well. Therefore, we keep a `HashMap` to get from the unchanged coordinates to the indices in the mesh.

3. Scale the node back down: as with step 1, we are just changing the coordinates, so the indices are unchanged.

Given that we can keep track of which vertices in the joined node are the ones we need to modify, we proceed to stitch it to the rest of the mesh so far:

1. Remove all faces connected with the straight cap center (in green in figure 23).

2. Remove the straight cap center itself.

3. Connect all possible ends with ends already existing in the mesh. The ones to be connected can be known if the Medial Axis tree is traversed depth-first: we know which segments of this node are connected to nodes that are already visited, these are the ones to be stitched back. In order to stitch back, we just need to find which are the indices of the accumulated mesh that coincide with the vertices that we want to connect. For this purpose, we keep another `HashMap` for the accumulated mesh, so we introduce the vertices to be stitched, get the corresponding indices, and add the new vertices and faces to the accumulated mesh, having in mind that the vertices to be stitched shouldn't be added, their existing indices have to be used instead.

All of this will solve the time growth problems, since we apply CSG operations to just a part of the total geometry each time, and we also solve the scale problem, since we scale up or down to a proper size for the CSG library to function properly. The only question now is how to know which one is the proper scale for each node.

5.2.3 Finding the proper scale factor for each joint

Calculating the scale factor for each joint is not easy, since it depends on the particular algorithm that the `J3DBool` library uses, and also the numerical properties of the lower level intersection computations used. Solving this problem analytically would require too much time, which would be better invested in approaching the problem in a different way, by not using this library. However, this would be too much for the scope of this thesis, so an alternative solution was developed.

The quick-and-easy solution is to keep trying scale sizes until one works. In a nutshell, the process will be algorithm 5.1:

Algorithm 5.1 Finding a working scale size

```
for all node ∈ nodes do
  repeat
    scale node with first unused scale size
    try CSG union for this node
  until CSG operation performs successfully
  merge joined node with the rest of the accumulated mesh
end for
```

The list of scale sizes to try is determined experimentally: centered in a value that is close to what usually works, it alternates going a bit bigger or smaller, so that it will eventually find a scale factor that doesn't break the CSG algorithm.

This process, although not the proper way of solving the problem, will nonetheless function accurately. Of course, this is only done this way because of time constraints, and would be dealt with in a different way, if the available time was longer.

5.3 Summary

The process of the generation of the mesh will consist of:

Algorithm 5.2 Overview of mesh generation

```
for all node ∈ nodes do
    calculate inner HCTCs (reference) that belong to it
    calculate outer HCTCs that belong to it
end for
for both thicknesses do
    for all node ∈ nodes, traversing depth-first do
        find scale factor
        scale up all HCTCs from this node
        perform CSG union of them
        scale the result down
        remove caps from ends of the node
        stitch the ends with any parent nodes
    end for
end for
subtract inner mesh from outer mesh
intersect with desired bounding box
```

6 Results

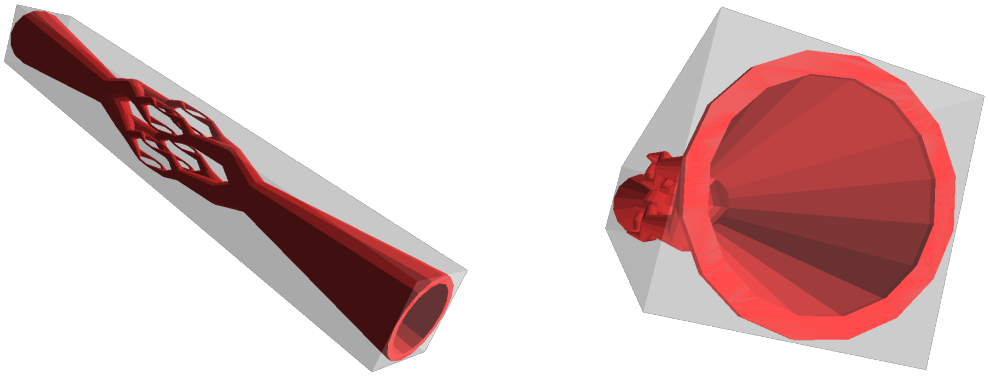


Figure 25: Screenshots from a mesh (in red) generated with the program. Includes the bounding box in transparent gray that is used as an input parameter to the program.

Using the input parameters given to us by the Laser Zentrum, we obtain the mesh seen in figure 25. Most of the mesh consists of the in and out tube, since there is where the needles will be connected to the vessel, in order to pump blood in and suck it out.

This is a test mesh, that will be used with the printer in order to check that the whole process pipeline works properly, and once the test is passed successfully, further meshes will be created with the program. Therefore, the input parameters are changeable through the interface.

The program features the generation of the tree given the parameters requested by the LZH, but by changing the process of the generation of the tree structure (as seen in section 4 in page 15), new meshes can be created with different shapes.

In figure 26, the variation on the general shape of the tree can be quite big. On the left, the number of levels N of the tree is increased. That can be done with the same input parameters, by reducing the thicknesses at the capillary ends, and reducing the maximum distance between them. On the right, a modification is done on the tree structure generation, by calculating the L-System parameters directly. In this case, the torsion at each bifurcation is modified with a random component, and the rest of the parameters are calculated by simple interpolation between the two ends.

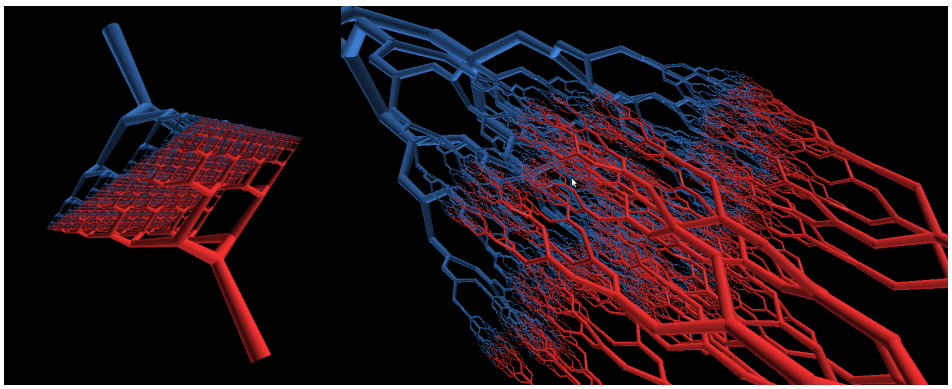


Figure 26: Alternative examples, by filling with a high number of levels of subdivision (left), and by adding some degree of randomnes on the torsion angle (right).

The Graphical User Interface (GUI) of the program consists of a panel with the controls, and a 3D view of the generated mesh (see figure 27). Parameters can be input manually, or saved to and loaded from a file.

Once the parameters are input, a preview of the mesh can be shown (see figure 28). This preview consists of the simple visualization of all the geometry together, without any merging or CSG joining done. The rasterization of the geometry as done by the 3D visualization makes it appear as if the mesh is already finished, but as seen on the right of the figure, the mesh is not hollow inside. Also, the ends are rounded, since they will be like that until the CSG intersection operation is done (see section 5.2 in page 26).

Despite not being the final result, it is a helpful feedback on the general shape of the mesh before it is generated. As an example, the preview for the LZH test mesh only takes about two seconds, while the final generation of the mesh takes more than two minutes. Given that with the input parameters themselves don't give an idea for the final appearance of the structure, the preview helps get quasi-immediate feedback for tweaking the parameters as desired.

Once the preview has been shown, the final mesh can be generated, and then saved as an STL file. For any of the two kinds of visualization (preview or final), some options are available: Stereo visualization (when possible, depending on the system), antialiasing (also depending on the system), black or white background, and transparency.

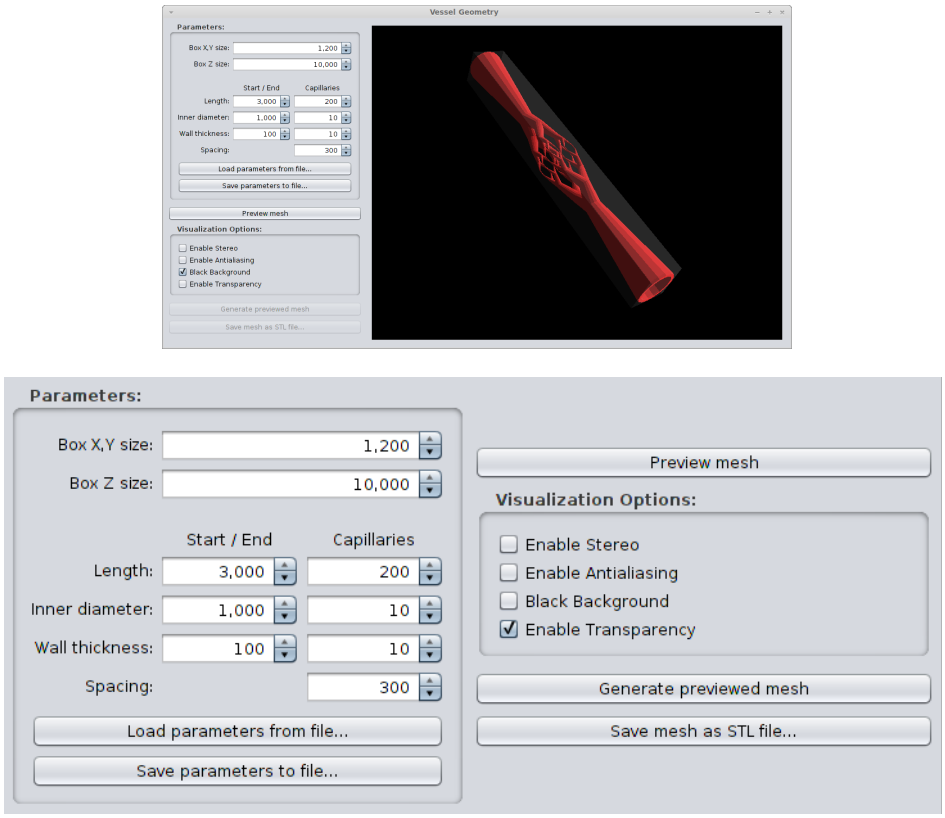


Figure 27: View of the GUI of the program: on top, the whole window, on the bottom, a closeup of the controls.

Enabling transparency can be helpful to examine the mesh and be able to see the thickness of the wall (see figure 29). With this, it can be confirmed that there are no self-intersections, that the wall thickness is properly interpolated from the beginning through the end, and that there are no occlusions in between both open ends.

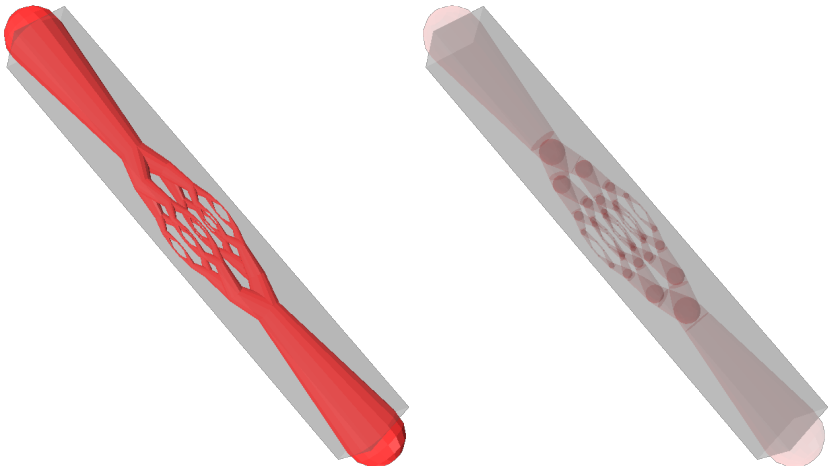


Figure 28: Preview of the mesh before finally generating.

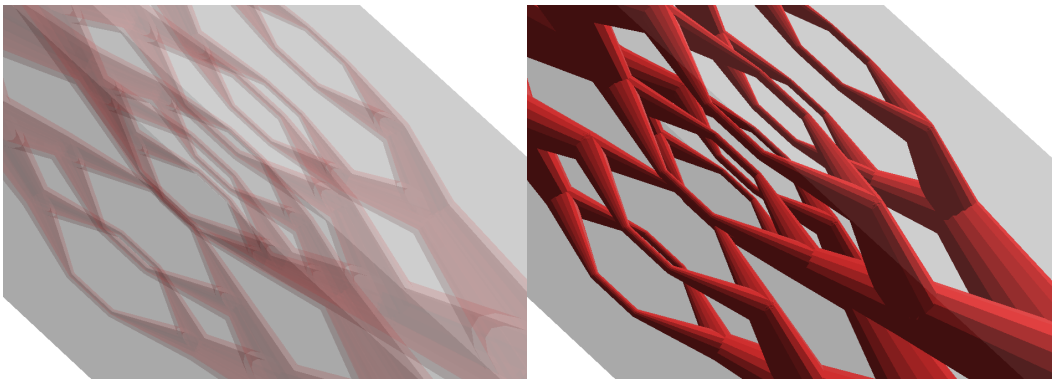


Figure 29: When exploring the result, transparency can be enabled.

7 Future Work

If the work done on this Thesis is to be followed, suggestions for future work are:

- Stop CSG usage. The CSG approach is valid in theory, since it solves the Medial Axis back-transformation problem, when dealing with non-triangulated surfaces and Medial Axes that are composed of linear segments. More specifically, it would solve the problem of joining all Medial Axis segments, whenever the Medial Axis back-transformation shape is computable for all of the segments. However, any implementation of CSG for triangulated meshes is going to have the same problems, mainly numerical problems when computing intersections between triangles.
- Research the back-transformation problem in itself. This was the hard problem of this thesis, and it poses some questions that would be interesting to research:
  - How to solve the problem at the intersections when the independent shape of each segment is known? This is the main problem encountered in this Thesis.
  - What would be the shape from a Medial Axis segment when it's curved instead of linear?
  - What would be the shape from a Medial Axis segment with a non-linear Radius Function?
  - How to deal with Medial Axis segments that consist of surfaces?
- Follow the colaboration with the Laser Zentrum. When the LZH test our final mesh and get some results from it, they might have some feedback on the shape of the mesh, or the quality of the triangles, etc. Furthermore, certain constraints like the regularity of the distribution of the capillaries, the angle of the bifurcations, etc. might have to be changed, since once the mesh is printed and blood is run through it, other non-geometric considerations (for example, flow requirements) might come to light, which will have to be addressed in further work.

8 Statement

With this signature I declare that I have written this research thesis all by myself and did not use any other but the quoted sources and materials.

Hannover, September 28, 2011

(Sergi Lazaro)

List of Figures

List of Algorithms

References

[1] Abelson, Harold ; DiSessa, Andrea A.: *Turtle geometry: The computer as a medium for exploring mathematics.* MIT Press, 1981 http://books.google.de/books?id=3geYp44hJVcC 14

[2] Blum, Harry: A Transformation for Extracting New Descriptors of Shape. In: *Models for the Perception of Speech and Visual Form* (1967), 362–380 11, 14

[3] Böttcher, Guido: *Medial Axis and Haptics (Diploma Thesis)*, Lehrstuhl für Graphische Datenverarbeitung, Gottfried Wilhelm Leibniz Universität Hannover, Diplomarbeit, 2004 14

[4] Colin Blakemore, Sheila J.: *The Oxford Companion to the Body.* Oxford University Press, 2001 7

[5] Laidlaw, David H. ; Trumbore, W. B. ; Hughes, John F.: Constructive solid geometry for polyhedral objects. In: *SIGGRAPH Comput. Graph.* 20 (1986), August, 161–170. http://dx.doi.org/10.1145/15886.15904 28

[6] Lindenmayer, Aristid: Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs. In: *Journal of Theoretical Biology* 18 (1968), März, Nr. 3, 300–315. http://dx.doi.org/10.1016/0022-5193(68)90080-5 7, 14

[7] Liu, Xuemei ; Liu, Huan ; Hao, Aimin ; Zhao, Qinping: Simulation of Blood Vessels for Surgery Simulators. In: *MVHI'10*, 2010, S. 377–380 4, 5, 14

[8] Prusinkiewicz, Przemyslaw ; Lindenmayer, Aristid: *The Algorithmic Beauty of Plants (The Virtual Laboratory).* Springer, 1991 http://algorithmicbotany.org/papers/abop/abop.lowquality.pdf. – ISBN 0387972978 14

[9] Wolter, F.-E.: Cut Locus and Medial Axis in Global Shape Interrogation and Representation. (1992) 14

[10] Wolter, F.-E. ; Friese, K.-I.: Local and global geometric methods for analysis, interrogation, reconstruction, modification and design of shape. In: *Computer Graphics International, 2000. Proceedings*, 2000 14

[11] Zamir, Mair: Arterial branching within the confines of fractal L-system formalism. In: *Journal of General Physiology* 118 (2001), 267–275 14